Università degli Studi di Padova
Dipartimento di Matematica "Tullio Levi-Civita"

CORSO DI DOTTORATO DI RICERCA IN: Mathematical Science
CURRICULUM: Computational Mathematics
CICLO: XXXVII

Tesi di Dottorato

Applications of Persistent Homology: Data Classification and Intrinsic Dimension of Manifold

Coordinatore
Ch.mo Prof. Giovanni Colombo

Supervisore
Ch.mo Prof. Stefano De Marchi

Dottoranda
Cinzia Bandiziol
N. di matricola: 2062305

Anno Accademico 2024/2025

La borsa di dottorato è stata cofinanziata con risorse del programma PON REACT-EU

To whom believes in me

List of Figures

List of Tables

Contents

Chapter 1

Introduction

The fourth industrial revolution, or better Industry 4.0, has linked the world of companies to the world of IoT (Internet of Things), which consists for example of Cloud technologies and predictive analysis of big data. Thus, cognitive technologies and, in particular, cognitive manufacturing have arisen that nowadays help companies by improving productivity, product reliability, quality, safety, and performance, minimizing downtime and reducing costs.

Therefore, it is clear how innovation and new technologies play an important role in making a country competitive in Europe as well as in the world. Research is crucial to face new challenges, and so the collaboration between companies and universities might be fruitful and successful. The present doctoral program fits well in this innovative background in the context of PON R & I 2014-2021. As partner of the project, the company *EnginSoft S.p.A* has provided the core of the collaboration: its own software `Smart Prod ACTIVE`. The thesis collects our analysis and results obtained by combining new theoretical techniques and the software reaching in this way the main goals of Strategia Nazionale si Specializzazione Intelligente (SNSI), which emphazizes how the collaboration has to bring an improvement of productive processes and a corresponding new organization of the production. Founded in 1984, EnginSoft is a multinational company of excellence in the field of Simulated Based Engineering and Science with about 4000 projects developed in Italy and Europe.

`https://www.enginsoft.com/it/`

Thanks to its own know-how and its highly qualified staff, it is today the ideal partner for supporting companies in favor of innovation and digital transformation. With a team of more than 250 qualified engineers with interdisciplinary skills in different fields and different technologies, EnginSoft is able to provide 360 degrees solutions starting from sale and customization of software to application and methodological consultancy, from high education to research. It is precisely the research a crucial point of the policy of the company, whose best motto, among the others, is

"It's impossible to bring innovation without research."

Thus, from its foundation, it has joined several research projects, one is presented in this thesis, going on with its journey towards innovation and evolution.

One of the most important solutions of the company is the software, named `Smart Prod ACTIVE`.

`https://www.enginsoft.com/solutions/smart-prodactive.html`

It is a web-based platform that brings companies into the era of data and artificial intelligence, allowing optimization in the production process and many other solutions. Thanks to it, data about a foundry process are available and have been studied with the aim of predicting the quality of a casting at the end of the process. More in detail, the task developed during the company collaboration was solving a classification problem by Support Vector Machine on highly imbalanced datasets. After a detailed analysis of this tool and thanks to the support of a tutor in the company, dr. Manolo Venturin, we began our mathematical study for applying new techniques and mathematical tools to Data Analysis, mainly to the analysis of industrial data.

Chapter 2 focuses on this software with a brief description of its structure and main modules.Then we have conducted a deep analysis of the available data on the foundry process with input and output or quality variables both. The last ones have been described in detail and then we show that they are independent, non-correlated. This observation makes it possible to manage each of them separately. So after solving 16 classification problems, the software will be able to assign a class of quality to the entire block, showing graphically the results in a suitable dashboard, determining if it is compliant or not regarding the global quality.

A promising and quiet new direction of study is the so-called Topological Data Analysis. Since it was born about twenty years ago and due to the fact that nowadays it is an active and popular field of research, it seems a good candidate to fit in our doctoral program. The reason why this remarked interest in such a field was derived from its strong and robust theoretical basis, for instance, from algebraic topology, and new information about the so-called *shape of data*, which is impossible to extract with other methods or techniques. Through its main tool, Persistent Homology (PH), it is able to find out and collect information regarding the topological or geometrical aspects that can characterize data in a more intrinsic and essential way. Another advantage of this theory is that PH is very transversal and versatile: it can be applied to, a priori, any kind of data, from point cloud data, to images, graphs, and finally to time series, only to name a few.

Chapter 3 offers a first insight into the world of *Topological Data Analysis*, shortly TDA, introducing the main definitions and concepts to make the theory of Persistent Homology more clear and understandable. Persistent features appear as points in $\mathbb{R}^2$, each point represents a characteristic with a recurrence, also greater than one. For this reason, such a collection of points takes the name of a multiset of points. Once computed, they are usually collected into some objects, easy to show graphically too, such as Persistence Diagram (PD) and Persistent Barcodes (PB). Finally, we report that a relevant property of this tool is its stability to noise [20], which is a significant aspect for applications to real-world data.

How to integrate these objects in a Machine Learning framework is not obvious, due to the fact that they are multisets. Performing statistics on these elements is also not straightforward. Thus in recent years, see for example [4], many different approaches have been developed to convert them into vectors of, in any case, making them treatable using Machine Learning tools. We recall only a few examples: **statistical vectorization methods** such as persistence statistics and summary of entropy, **algebraic vectorization** as tropical coordinate functions, Betti curves, lifespan curve, persistence landscape, persistence silhouette, PI ([2]) and **kernel approaches** (see [55], [39], [16], [41], [23], [10]). In general, some references can be found about connections between Machine Learning and TDA [44], [18], [34], [47] or comparisons between methods of TDA [7], [53]. In the literature some approaches have been proposed also in the direction of Deep Learning ([33], [61]), creating suitable topological layers to insert into prexisting architectures of Neural Networks in order to be able to incorporate topological tool. To our knowledge, the most common are Perslay [17] and Pllay [37].

In Chapter 4, we first analyze a method that uses the theory of TDA incorporating a suitable modification of a famous baseline method for classification, the Kth Nearest Neighbors (KNN). We study the method proposed in [38] showing also drawbacks and classification performances in a comparison between baseline methods such as the Support Vector Machine (SVM) and Decisional Tree (DT). We also run tests on several datasets and on data provided by the company, too. Then another interesting topic has been taken into account: the definition of suitable kernels in the SVM framework to classify PDs. In fact, since the space of PDs is only metric, to use methods that require data to live in a Hilbert space, such as SVM and PCA, it is necessary to introduce the notion of kernel. In this context, we refer to Persistence Kernels (PKs) that are able to map PD to space with more structure, where it is possible to apply techniques that need a proper definition of inner product.

Each kernel depends on some parameters that needed to be tuned. So we lead the so-called *shape parameter analysis*, as in the context of kernel for interpolation, which, as far as we know, is totally absent in the literature. Once we have pointed out the best parameter values and thanks to the transversal nature of PH, we have compared performances using different kernels in classifying PDs from different kinds of data. Finally, after adjusting the values for the parameters, we have reported our results on the classification performance of the PKs on different types of data.

Another relevant application of PH is in the context of *Intrinsic Dimension*. In the Machine Learning framework, the user usually deals with a large amount of data with a huge number of features. It is clear that the more features a dataset has, the more difficult it is to treat and analyze it. However, tests and real applications point out that only a few of these features are indeed significant and essential to represent the whole dataset. So, the number of these essential features is exactly the idea behind the concept of Intrinsic Dimension (ID). Hence, it clarifies the efforts made by researchers in the field to discover a new method to estimate the ID and allow for dimensionality reduction, making data easier to analyze. Chapter 5 is completely dedicated to this topic. First, it outlines the most popular methods available today to estimate ID, then shows definitions of ID, such as the box counting dimension and correlation dimension, and finally it points out how PH can be used to create ID estimators, describing **i-Dimensional Persistent Homology Fractal Dimension**, **Persistent Homology Dimension** and **Persitent Homology Complexity**. To check the goodness of these ID estimators, we have run tests on datasets with different nature: benchmark datasets, for instance, points for regular surfaces, fractal and attractors and data from neuroscience. Also in this context, we first provide a shape parameter analysis, then we check how the estimators are able to approximate ID of common datasets and finally we apply them to data taken from the world of Neuroscience.

Chapter 2

SMART Prod Active

The expression 4th Industrial Revolution or better Industry 4.0 was coined during the exhibition in Hannover, Germany, in 2011 and today is continuously enforced by technological and innovative developments of companies all over the world.

The term Industry 4.0 aims at outline today's tendency of industrial automation to insert and integrate new productive technologies with the purpose of increasing productivity, improving the quality of final products, creating new business and improving work conditions. Actually, its realization requires a probably long innovation process in many companies, but in the end it brings them closer to the condition of a fully integrated company with the final aim of making industrial processes more efficient and companies more competitive. This can be achieved through a mix of technology in robotics, sensor science, automation, information, connection, and programming.

Through the so-called 'digital transformation', all become smart, the production and distribution themselves are now SMART, which means smarter, faster, and more efficient. Smart Production, new productive technology that interacts with all aspects of production allowing partnership between men, machines, and systems, Smart Services, new management of IT infrastructures, Smart Energy, new supply systems and a careful monitoring of energy consumption making the infrastructures more eco-friendly and high-performance, Smart Lifecycle management, which includes the development process of each new product, Smart Supply Chain, that involves planning of financial and physical flux in the whole supply chain, and finally the Smart Factory, that encloses the entire management related to infrastructure, services, among all logistics, maintenance, quality and safety. We want to focus now exactly on this new concept of Smart Factory. Among the technologies used, it includes:

- **Cloud connectivity**: all data and information from Smart Factory have been directed to it. Its connectivity ensures that each department works on real-time data and it can observe immediately all systems within the supply chain
- **Artificial Intelligence (AI)**: exploiting AI in systems makes all processes faster, more powerful and, flexible
- **Machine Learning**: it is useful for Smart Factory because it is able to make predictive maintenance, and monitoring processing to prevent problems and to reduce downtime
- **Big Data**: a big amount of data that allows advanced and predictive analysis
- **Industrial Internet of Things (IIoT)**: all devices and machines, that can send and receive digital data, build the IIoT net
- **Digital twin**: it produces exactly the virtual mirror of a machine or a system allowing analysis and modeling with no risks
- **3D printing**: it allows to exploit smart automation for on-demand production
- **Virtual (VR) and Augmented Reality (AR)**: thanks to real-time data, it allows to obtain a summary of the global situation of the company with no limitations
- **Blockchain**: it can be used in the creation of "smart agreement" with providers, the monitoring of goods sources and, management along the whole supply chain
- **New generation databases**: they represent the brain that moves Industry 4.0 and the Smart Factory at all

Thanks to its technologies, the Smart Factory is able to:

- **Acquire data**: through AI and new databases, it is now possible to collect a useful set of data about the whole company, the supply chain and the external world. Using sensors and getaway, the IIoT allows all connected machines to send measurements to the system.
- **Analyze data**: the ML and smart systems exploit the advanced analysis and modern solution for data management with the aim of giving meaning to them. The IIoT sensors are able to alert when machines need fixings and maintenance

- **Automate processes**: once the first two steps have done, the new work flux have been defined and new instructions have been send to machines and devices within the system.

This innovative approach to industry, of course, has some possible advantages. The attention is focused on predictions of problems or defect instead waiting for the malfunctions to appear, it is easier to apply more eco-friendly and safe industrial policy, monitoring in a faultlessly manner sources and quality of each raw material in the whole supply chain. Finally, advanced analysis of system data quickly reveals weak points and areas of improvement.

With the allocation of doctoral programs related to PON R&I, Italy encourages and supports the fourth industrial revolution, creating a link between the University and the world of industry. In this way, the collaboration among them is crucial, since University may exhibit the last findings of scientific research, and, on the other side, industry can offer a concrete and real problem to address. This partnership reveals convenience for both: The universities have been encouraged to face new challenges, and the industrial world can improve its processes, becoming always more and more efficient and competitive. Industry 4.0 has involved transversally all industrial areas, including that of metal and foundry. This is in fact the context where we have worked during the phd, focusing mainly on Green aspects, such as reducing waste, non-compliant products, downtime and making the production more eco-friendly. An example of technology used in the Smart Factory is the previously mentioned IoT. It can be used in different contexts as:

- Apply to quality monitoring
- Predict fails and downtime
- Control the work progress of the production by means of machine-to-machine connection.

In it indeed on the first of these fields that we are interested in and in this context, Smart Prod Active fits really well. In fact, thanks to its modules, it is able to carry out in many areas of interest: from the acquisition and monitoring of data to the analysis of data for predictive maintenance, from energy consumption monitoring to quality one. All information about software is available on the website

`https://www.enginsoft.com/solutions/smart-prodactive.html`

More in detail, it consists of 4 modules, that are:

- **smartprodactive.a** [acquisition and monitoring]: it is an innovative solution for acquisition, recorder and monitoring of industrial data coming from machines, sensors, ecc . . . Once data have been historicized, they allow a real time check of productive process, analysis of trend, ecc . . .

- **smartprodactive.m** [maintenance]: its task is to increase efficiency of industrial machines through the so called predictive maintenance. Data have been processed with an advanced algorithm of AI to find anomalies and prevent potential failures. Thus, it allows companies to decrease the maintenance costs and maximize productivity, reducing downtime.

- **smartprodactive.e** [energy]: with data related to energy consumption, this module is able to intercept trend and notify anomalies related to consumption, allowing the client company to reduce environmental impact in favor of greater sustainability.

- **smartprodactive.q** [quality]: thanks to AI, it allows to identify quality problems and define improvement plans. This module guarantees processes to be efficient, effective, and that reach the required level of quality. The main aim is the reduction of waste, the guarantee of quality, and the minimization of non-compliant products. The results related to quality can be visualized graphically in report, plots and dedicated dashboard.

The thesis work focused on this last module, about *quality control.*

In a context that is always more competitive than that of the manufacturing industry, the quality of the product is essential. It involves monitoring, measuring, and evaluating the quality of output throughout the production process. In addition, it helps companies remain competitive by improving consistency, reducing costs, enhancing customer satisfaction, and maintaining a strong brand reputation. Furthermore, it plays a key role in driving efficiency, innovation, and risk management, making it indispensable for sustainable business success in any industry. This is the reason why many companies today invest a lot of money in this direction. In our investigation, we are mainly interested in application of control quality to foundry of aluminum.

Foundry is generally a factory where the casting process is carried out to meet customer requirements or vending targets and using different metals or raw materials such as aluminum, cast iron, bronze, brass, steel, magnesium, and zinc. Casting is a process of melting metal, pouring it to mould after the necessary heat treatment, and allowing the metal to solidify to take the desired shape. The end product of a foudry is generally called *casting*. More in detail, the foundry process typically involves several key phases.

1. **Pattern Making**: This is the first step where a pattern of the desired object is created, usually from materials like wood, metal, or plastic. The pattern is used to form the mold.

2. **Mold Making**: In this phase, the pattern is used to create a mold. The mold can be made from sand, metal, or other materials, depending on the casting process being used.

3. **Melting**: The metal that will be cast is melted in a furnace. The type of furnace and the temperature depend on the metal being used.

4. **Pouring**: Once the metal is molten, it is poured into the mold. This step requires precision to ensure that the mold is filled correctly without any defects.

5. **Cooling**: After pouring, the metal needs to cool and solidify. The cooling time can vary based on the type of metal and the thickness of the casting.

6. **Mold Removal**: Once the metal has cooled and solidified, the mold is removed to reveal the cast object.

7. **Finishing**: The final phase involves cleaning, machining, and finishing the cast object to meet the required specifications and surface quality.

In Industry 4.0 the global process is usually taken under control using dedicated software. For what our analysis concerns, we have investigated a particular software, `SMART Prod Active`, that can give benefits in two manners:

- for **Monitoring**: it provides real-time data about the process under exam and information about quality

- for **Prediction**: it runs a script and makes prediction using a selected and integrated model

The final goal of our project is a model or classifier that, based on data acquired, is able to predict the quality of a casting. Once created in Python, the model has to be converted into a binary .p file, supporting hence the integration in the preexisting infrastructure.

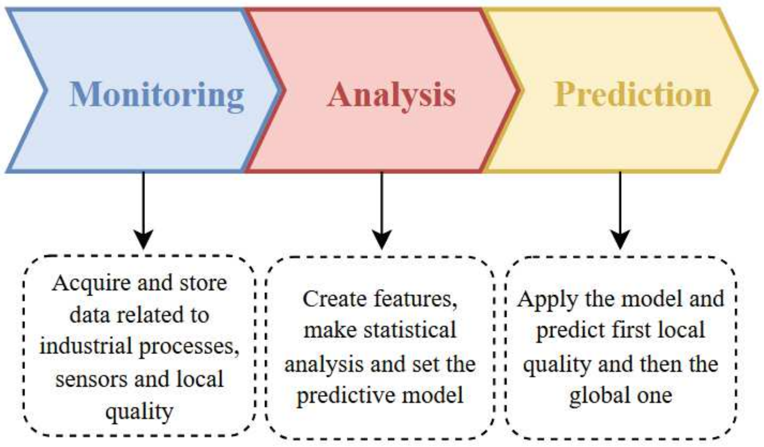

Monitoring
Analysis
Prediction
Acquire and store data related to industrial processes, sensors and local quality
Create features, make statistical analysis and set the predictive model
Apply the model and predict first local quality and then the global one

Figure 2.1: Process workflow

In Figure 2.1, we represent clearly our final goal: create a new intermediate box, between those provided by Smart Prod Active, that may connect them each other.

More precisely, the process globally should work in the following manner:

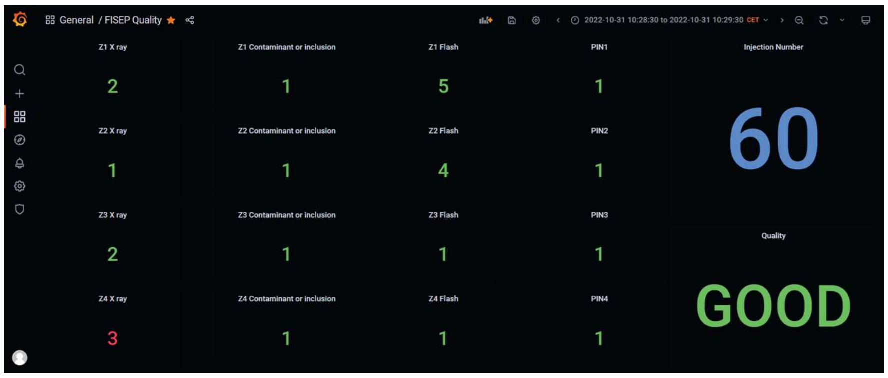
1. with its Monitoring power, Smart Prod Active can acquire a lot of data related to industrial processes, sensors and local quality (**Monitoring**)

2. the Analysis Box takes as input the data of the previous step, creates features, make statistical analysis and set the predictive model (**Analysis**)

3. the Predictive box finally applies the model to predict first local quality of casting and then the global one. The obtained results are then made user-friendly to show using dedicated prexisting dashboard. (**Prediction**)

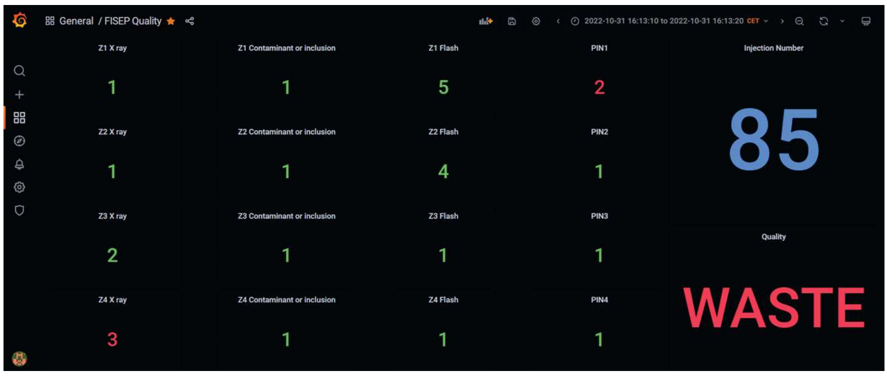
Dashboard exhibits and collects many information as the value of quality for each variable, the injection number or identification number of the job under analysis in blue and finally a whole quality index. If the Quality value is **GOOD**, see Figure 2.2, the casting is compliant while if it is **WASTE**, see Figure 2.3, it doesn't reach the expected level of quality and thus it must be excluded and removed from the final production.

In the following section, we will explore deeply the second phase **Analysis** focusing on data, their features and statistical analysis of them.


General / FISEP Quality
2022-10-31 10:28:30 to 2022-10-31 10:29:30 CET

Z1 X ray 2
Z1 Contaminant or inclusion 1
Z1 Flash 5
PIN1 1
Z2 X ray 1
Z2 Contaminant or inclusion 1
Z2 Flash 4
PIN2 1
Z3 X ray 2
Z3 Contaminant or inclusion 1
Z3 Flash 1
PIN3 1
Z4 X ray 3
Z4 Contaminant or inclusion 1
Z4 Flash 1
PIN4 1
Injection Number 60
Quality GOOD

Figure 2.2: Dashboard 1


General / FISEP Quality
2022-10-31 16:13:10 to 2022-10-31 16:13:20 CET

Z1 X ray 1
Z1 Contaminant or inclusion 1
Z1 Flash 5
PIN1 2
Z2 X ray 1
Z2 Contaminant or inclusion 1
Z2 Flash 4
PIN2 1
Z3 X ray 2
Z3 Contaminant or inclusion 1
Z3 Flash 1
PIN3 1
Z4 X ray 3
Z4 Contaminant or inclusion 1
Z4 Flash 1
PIN4 1
Injection Number 85
Quality WASTE

Figure 2.3: Dashboard 2

2.1 Dataset: features and output variables

We want to study a foundry process for aluminum. During its entire job, a plunger goes through three phases:

1. it injects the lega of molten metal expelling the air inside the mold
2. it compacts the material
3. it stabilize and comes back to its initial position

During an entire job, some quantities have been measured and stored to be analyzed later. This study is indeed interesting from an industrial point of view, since it allow to make statistics about global quality of the production taking also under control number of fail and waste products, guaranteeing high quality of the chain of production and, eventually, applying corrective strategies to fix problems.

The data available can be divided into three groups:

- **Process parameters**
- **Features from sensors and thermocameras**
- **Quality** or **Output Variables**

Process parameters come from a Design Of Experiment (DOE), made by a partner of Enginsoft during their collaboration. They are the main topics of a relevant investigation tool in the industrial and manufacturing world the **DOE**.

Design of experiments **DOE**, also known as experiment design or **experimental design**, is a systematic, efficient method that enables scientists and engineers to study the relationship between multiple **input variables** (**process variables** or aka factors) and key **output variables** (aka responses). It is a structured approach for collecting data and making discoveries where statistic plays an important role. It is widely considered that DOE (or experimental design) forms an essential part of the quest for effective improvement in process performance or product/service quality. Some manufacturing problems that can be addressed using an experimental approach include development of new products; improvement of existing processes or products; improvement of the process/product performance relative to the needs and demands of customers; reduction of existing process spread, which leads to poor capability. In out context, these variables are related to: the temperature of the oven, the initial and mean speed, the mean pressure, time of spray and jet volume of the entire system. During DOE, all data are collected for each job, and some information have been added to each of them as some keys `ID, QR, AcqTTV,` the number of `Injection`, the number assigned by the software `ProdACTIVE`, the time of the experiment in `InjectionTime` and the number of the `Design`. These data derived from experiments run in two days: on 1^{st} November and on 1^{st} December 2022. They have been measured and stored using the software `SMART Prod Active`.

Concerning the features of **sensors and thermocameras**, they have been calculated / extracted from quantities measured by `SMART Prod Active` during each experiment.

They are time-dependent quantities and thus look like time series. They are stored into .csv files. Managing these types of raw data is useful in convert them into something easier to treat, the *features*. **Feature engineering**, in data science, refers to manipulation, addition, deletion, combination, mutation — of your

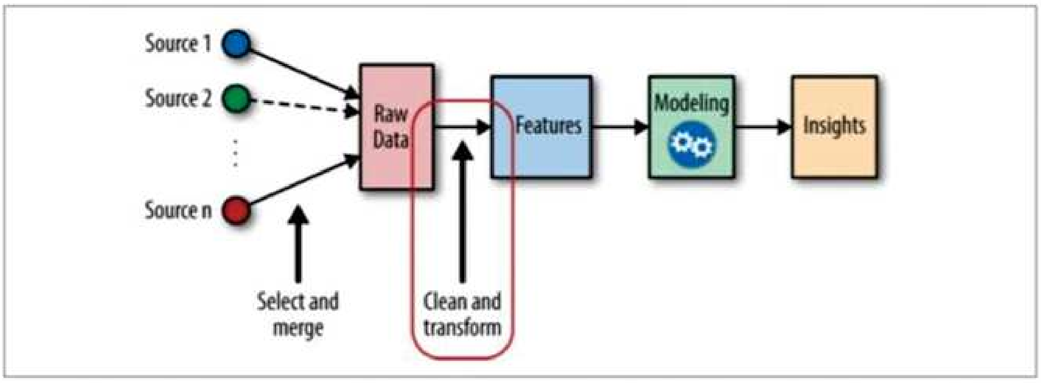


Figure 2.4: Standard workflow with in red the phase of Feature Engineering

data set to improve machine learning model training, leading to better performance and greater accuracy. Starting from the time series related to the position of the plunger, it was allowed to compute the corresponding time series about speed and acceleration to store together with data coming from 7 sensors of temperature and 2 of pressure. From all these time series, finally some features have been extracted: starting time of different phases, maximum and minimum values during the process about speed, acceleration, temperature, head pressure, counter pressure, time at which these extreme quantities have been measured and some integral quantity, suggested by experts in this context, as:

- **Root mean square of the acceleration** $[m/s^2]$,

$$a_{RMS} = \sqrt{\frac{\int_{t_{s2}}^{t_{e2}} \ddot{x}(t)^2 dt}{t_{e2} - t_{s2}}}$$

- **Work of the plunger** $[bar * m]$

$$Li = \int_{t_{e2}}^{t_{e3}} p(t)\dot{x}(t)dt$$

  where p denotes the pressure

- **Energy of flux forces** $[m^3/s^2]$

$$E_f = \int_0^{t_{e2}} \dot{x}(t)^3 dt$$

About **quality variables**, helped by experts in the foundry process, four more significant zones have been detected in the final block. Figure 2.5 and Figure 2.6 depict Zones 1 and 2 and Zones 3 and 4, respectively.

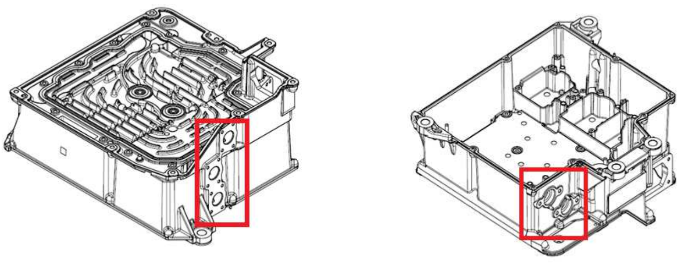


Figure 2.5: Zone 1 and Zone 2

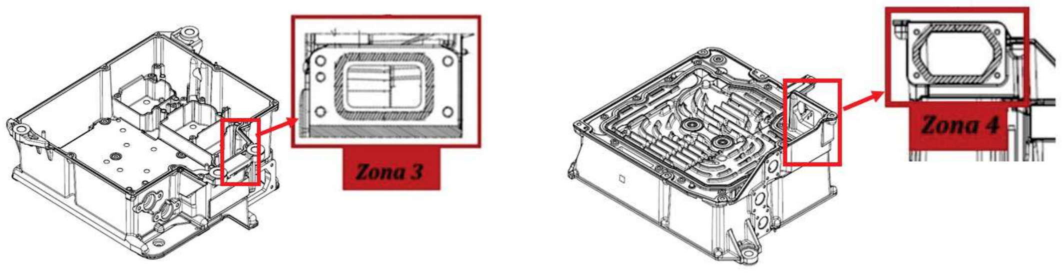


Figure 2.6: Zone 3 and Zone 4

In these zones the local quality have been measured considering: presence of contaminant or inclusions `Z_n_1`, Flash `Z_n_2`, X Ray `Z_n_3` with $n = 1, 2, 3, 4$. Then, we also have the quality measurements about 4 PINS, as in Figure 2.7.

Finally, we end up with 16 output variables that take values in $\{1, 2, 3, 4, 5\}$, where 1 denotes the optimal quality and 5 the worst.

All the aforementioned data are stored in an Excel file but through the key `ProdActive` provided by the software, the merge between files is possible. The final dataset consists of about 300 rows, one for each repetition of the foundry process, about 230 **variables**, process variables and sensors, and 16 **output variables** or **variables of quality**.

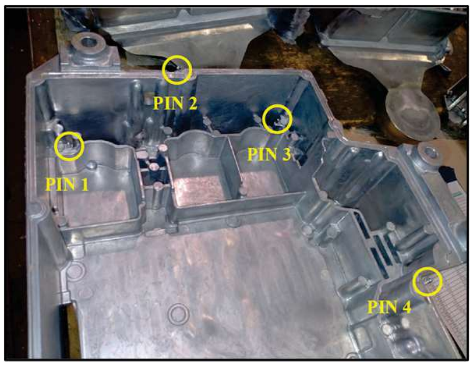


Figure 2.7: 4 PINs into the casting

2.2 Statistical analysis of data

Dealing with data coming from real-life processes is not straightforward, and some challenges have to be faced. First of all, the main elements of descriptive statistics as count, mean, standard deviation and quartiles have allowed the identification and consequently the elimination of outliers that will not be taken into account for future analysis. Furthermore, lines with missing values have been removed.

Then, a global correlation analysis have been done to reduce the number of variables for the model. A particular attention has been pointed towards output variables and their correlation. First, we have investigated if some of them could be useless because constant. Unfortunately, this is true for some of them, `Z_2_1`, `Z_3_2`, `Z_4_2` and `PIN_4`, that have been deleted.

The analysis of correlation between output variables is another useful step. If some variables would turn out to be correlated, one should remove some of them because encode more or less the same information and so they are redundant and so useless. In statistic there are many ways to reach the goal but probably the most common choice is the **Pearson correlation coefficient (PCC)** that measures linear correlation between two sets of data. Mathematically speaking, given two random variables X, Y, their PCC is defined as

$$\rho(X, Y) = \frac{cov(X, Y)}{\sigma_X \cdot \sigma_Y}$$

where **cov** denotes the covariance and σ_X, σ_Y are the standard deviation of X, Y respectively.

When applied to samples, the so-called sample Pearson correlation coefficient r_{xy}, if we have n couples of points $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, is given by

$$r_{xy} := \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{j=1}^{n}(x_j - \bar{x})^2}\sqrt{\sum_{j=1}^{n}(y_j - \bar{y})^2}}$$

where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the sample mean. Analogously for $\bar{y}$. The correlation coefficient ranges from -1 to 1. If it is exactly equal to 1 or -1 implies that a linear equation describes the relationship between samples from X and Y perfectly, with all data points lying on a line. In our analysis, we have considered two variables correlated if $r_{xy} \in [-1, -0.9] \cup [0.9, 1]$. For each pair of variables, the Pearson coefficient has been computed and all such coefficients have been shown graphically in the correlation matrix as in Figure 2.8.

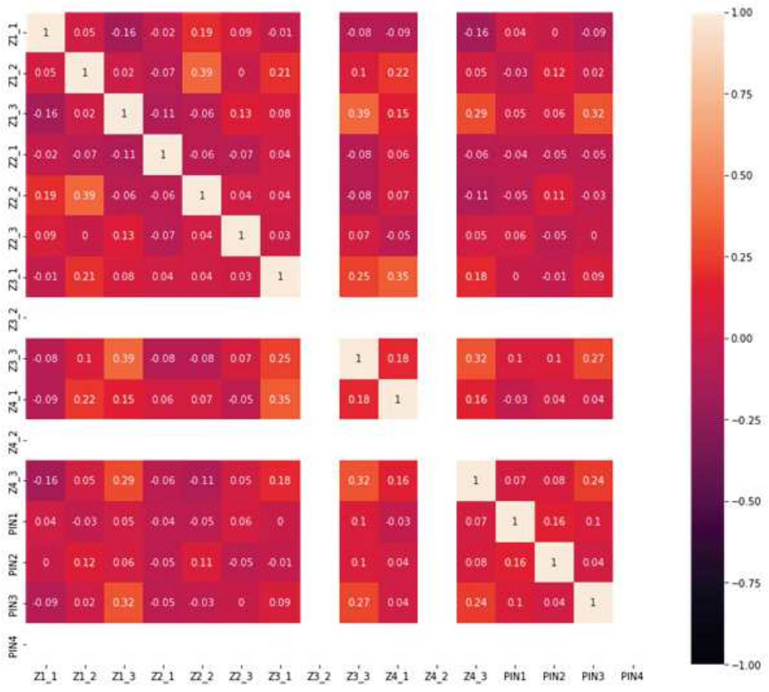


Figure 2.8: Correlation matrix of output variables

For the sake of completeness, we have considered all 16 output variables here and not only the 12 more significant. It can be highlighted that the output variables are not correlated and thus they can be treated singularly. Furthermore, it means that an error in one zone of the block doesn't propagate inevitably to errors into other zones on the casting.

The non correlation between quality variables results in the application of the selected models, like SVM, KNN, DT, NN, 12 times, 1 for each output variable to predict. However, given an output variable, it is reasonable that not all input variables have the same impact in determining its output value.

Therefore, for each of 12 quality variables, using `permutation_importance` Python function applied to the standard KNN classifier, the 10 main significant features have been detected. This importance can be evaluated by measuring how the accuracy of the model changes if this variable is not available. The number 10 has been selected after several tests and in the end it revealed a good compromise. This choice aims to avoid the model to focus on local minima that cause the so-called overfitting. So, for each output variable, the model will take as input only the 10 meaningful variables to make the prediction.

But another interesting aspect of data from real-life phenomenon needs our attention: the possible imbalance nature of data.

2.3 Data augmentation with SMOTE

Another aspect to take into account is that the data could be incomplete or also **highly imbalanced**, for further details about this see the corresponding section in the appendix. After the previous statistical analysis, now we have to deal with 12 output variables and 5 are admissible classes. For each of them, we report in Table 2.1 its distribution in classes where columns `Class N` contain the number of samples per class N.

From Table 2.1, one can see very well all issues that is common to meet when handling with real life data. First, we don't have samples for each class and so it is impossible to be able to predict such classes for that specific variable. For example, `PIN2` have no representatives for class 4 and 5. In such case, no method can be used to fix the problem since data are totally absent and therefore one can't claim to classify into 5 classes but only in 3. Similarly, if samples of a class are too few the situation is similar to the previous one, with no points in the class. If we consider `Z2_2`, we have only 1 representative for class 1. Obviously, the information is not enough to make possible the prediction of points with that label. The consequence is that these few samples will be removed and the classifier will be set to predict only a subset of labels.

Fortunately, in cases where the samples are small in number but not too much, something can be done to increase conscientiously the number of samples. This process takes the name of **Data Augmentation**. Real-world data is often limited or hard or expensive to collect, so the dataset results are small and difficult to manage.

Variable	Class 1	Class 2	Class 3	Class 4	Class 5
Z1_1	56	157	65	11	2
Z1_2	140	102	24	9	16
Z1_3	63	201	27	0	0
Z2_2	1	188	57	26	19
Z2_3	184	107	0	0	0
Z3_1	89	184	18	0	0
Z3_3	54	170	58	9	0
Z4_1	22	168	90	11	0
Z4_3	70	207	14	0	0
PIN1	254	36	1	0	0
PIN2	223	59	9	0	0
PIN3	225	35	19	10	2

Table 2.1: Distribution of foundry data per class

This is only a motivation for the widespread use and fortune of this approach. In Machine Learning, these corresponding methods are also known as Oversampling techniques, see the Appendix for a deep overview.

Basically, Data Augmentation is a technique used in Machine Learning and Deep Learning to increase the diversity of your training dataset without actually collecting new data. It involves creating modified versions of existing data points through various transformations. For example, in image processing, you might rotate, flip, crop, or change the brightness of images to create new training examples. This helps improve the model's ability to generalize and perform better on unseen data by exposing it to a wider range of scenarios. Data augmentation is especially useful when you have a limited dataset, as it can help prevent overfitting and enhance the robustness of your model.

Currently, many different approaches are available in the literature, among all **SMOTE**. It is a data augmentation technique that uses the KNN algorithm to produce new synthetic data that reduce the imbalance of the entire dataset. For further details, see dedicated section in the Appendix. We have decided to use SMOTE since it is easy, effective, and very common in engineering contexts.

Unfortunately, there is no standard recipe that can guide the application of the method. Obviously it depends on the problem to address and the kind of data to handle. With only two classes the application seems to be obvious, but with more than two labels the situation can become harder. As we have previously mentioned, it is not known a priori how many samples are generated for minor classes. Without any doubts, SMOTE may build as new samples as the user desires, but one has to be careful not to exaggerate. For example, in the case of 3 classes with 100, 98 and 5 samples, respectively, SMOTE can build points to reach a new dataset with 100, 98 and 100 representatives. However, it is also clear how inappropriate use of the method can bring to an artificial biased model that is unable to generalize well onto real new data. The only possible rule and suggestion to follow is the use of SMOTE to increase the number of samples conscientiously.

After this preliminary discussion, we remanage data deleting classes with no or almost no samples, considering only quality variables with samples in at least 3 classes among 5 (remove `Z2_3` and `PIN1` variables) and applying SMOTE. Finally, we end up with a set of quality variables as described in Table 2.2 where columns `N. Cl.` denotes the number of classes that can be considered for that specific variable, `Class N` the number of samples per class after using SMOTE and `IR` is the related Imbalanced Ratio that measures how imbalance the dataset is (See Appendix for more details). SMOTE has been performed using its implementation in `imblearn.over_sampling` [77], setting up `k_neighbors` equal to 3.

Variable	N. Cl.	Class 1	Class 2	Class 3	Class 4	Class 5	IR
Z1_1	4	56	157	65	50	**0**	3.14
Z1_2	5	140	102	50	50	50	2.8
Z1_3	3	63	150	50	**0**	**0**	3
Z2_2	4	**0**	188	57	50	50	3.76
Z3_1	3	90	184	50	**0**	**0**	3.68
Z3_3	4	50	150	50	20	**0**	7.5
Z4_1	4	50	168	100	50	**0**	3.36
Z4_3	3	70	150	50	**0**	**0**	3
PIN2	3	200	90	50	**0**	**0**	4
PIN3	3	150	40	40	40	**0**	3.75

Table 2.2: Final distribution of quality variables

Chapter 3

Topological Data Analysis and Persistent Homology

In the last two decades, with the increasing need of analyzing big amount of data, that usually are complex and of high dimension, it was revealed meaningful and helpful to discover new and further methodologies in order to provide new information from data. This has brought the birth of Topological Data Analysis, shortly TDA, is an emerging discipline seeking to examine geometric properties of data using tools from algebraic topology [15]. Its main aim is to extract intrinsic, topological features from data, related to the so-called *"shape of data"*. These kinds of features, collected in the Persistence Diagrams, has been revealed winning in many different applications, mainly related to applied science, improving the performances of models and of classifiers, as in our context. Thanks to the strong theoretical basis behind, the TDA is very versatile and can be applied to data with a priori any kind of structure, as we will explain below. Thus it has found a lot of different applications as chemistry [65], medicine [5], oncology [11, 46], biomedicine [60], neuroscience [48, 28, 8], finance [45] and computer graphics [9] only to name a few.

The main tool of TDA is the so-called Persistent Homology, which allows us to extract persistent topological features from data. This method derived directly from the Algebraic Topology, that is a branch of math that uses tools from linear algebra to study topological spaces in order to find its algebraic invariants. An example of such invariants are the homology groups. Intuitively, given a topological space X, the n Homology Group, $H_n(X)$, consists of the $n-$dimensional holes that characterize the space itself. In practical application, users usually consider only 0,1,2 dimensional holes as we will explained later.

From a general point of view, if we have a topological space X, that is given by a couple (X, τ) where X is a set and τ a topology on it (for example a sphere), the aim is to count the number of connected components (0-dimensional holes), cycles (1-dimensional holes) and cavities/voids (2-dimensional holes) with the idea that these numbers can indeed represent and characterize the space X from a qualitative and intrinsic point of view.

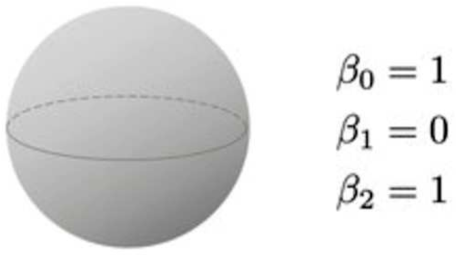


Figure 3.1: Sphere with its Betti Numbers

The numbers reported near the figure represent such holes, the so-called *Betti numbers*, that will be defined properly in the following Definition 10. For instance, the rank of the homology groups previously mentioned. In the case of the sphere, it is evident how there is only 1-connected component, 1-cycles, and 1-cavity inside the sphere.

The idea is to understand how to extend this theory for dealing with discrete data as, for example, point clouds. To reach the purpose, it is needed to put some geometrical structure into data and this can be done thanks to the theory of *Simplicial Homology*. This theory consists of applying homology to structures known as simplicial complexes, which are the generalizations of triangulation of a topological space. Then, starting from a discrete dataset, as a point cloud, the idea is to consider not only one simplicial complex built upon points, but a nested sequence of them, always more and more complex, and see which topological features appear and disappear through the evolution. This is the key idea of Persistent Homology.

Before going into the description of the topic, we need to give a brief introduction to simplicial homology.

3.1 Simplicial Homology

The field of Algebraic Topology aims at studying topological spaces through tools from abstract Algebra. It can classify topological spaces based on algebraic invariants. Some of them are the Homology Groups and to build them in case of varieties, one can turn to structures similar to triangulation, known as Simplicial Complexes. Roughly speaking, these structures consist of gluing together simplices of different dimensions according to some rules.

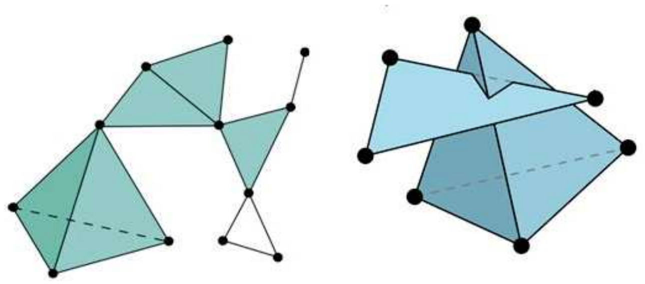


Figure 3.2: An example of a valid simplicial complex (left) and an invalid one (right)

More precisely,

Definition 1. A **simplicial complex** $\mathcal{K}$ consists of a set of simplices of different dimensions and has to meet the following conditions:

- Every face of a simplex σ in $\mathcal{K}$ must belong to $\mathcal{K}$
- The non-empty intersection of any two simplices $\sigma_1, \sigma_2 \in \mathcal{K}$ is a face of both σ_1 and σ_2

The dimension of $\mathcal{K}$ is the maximum dimension of simplices that belong to $\mathcal{K}$. In this section, we denote with n its dimension.

We show in Figure 3.3 examples of simplices of low dimensions, that are of dimension 0 (vertices), dimension 1 (edges), dimension 2 (triangles), dimension 3 (tetrahedron), and so on. Given a simplex σ of certain dimension n, it is completely defined by its set of vertices and so denoted by $[v_1, \ldots, v_{n+1}]$. Every subset ρ of $\{v_1, \ldots, v_{n+1}\}$ represents another simplex but mainly a *face* of σ, briefly denoted by $\rho \leq \sigma$.

Thus we can divide simplices in $\mathcal{K}$ into groups based on their dimension k and we can enumerate them using Δ_i^k. If $G = (\mathbb{Z}, +)$ is the well-known Abelian group, we may build linear combinations of simplices with coefficients in G, and so we introduce the following

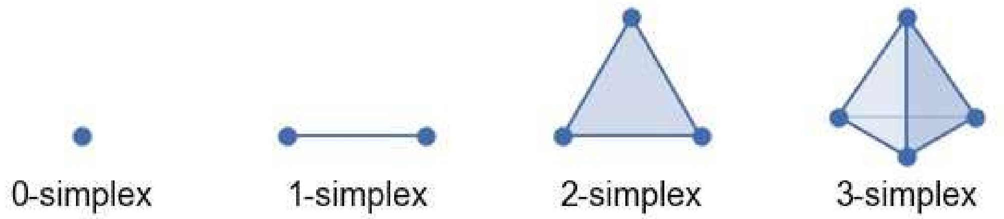


Figure 3.3: Examples of simplices of dimension $0, 1, 2, 3$

Definition 2. A **integer valued k-dimensional chain** is an object of the form

$$c = \sum_i a_i \Delta_i^k$$

with $a_i \in \mathbb{Z}$.

The simplest examples of chains are of the form Δ_i^k and $-\Delta_i^k$. If we define the sum of two chains as the chain with coefficients the sum of the coefficients of the initial chains respectively, then the set of chains becomes an Abelian group $C_k(\mathcal{K})$.

Definition 3. The set S of integer-valued k-dimensional chains endowed with the binary operation

$$+ : S \times S \to S$$

defined $\forall c_1, c_2 \in S$ as

$$c_1 + c_2 = \sum_i a_i \Delta_i^k + \sum_j b_j \Delta_j^k := \sum_l (a_l + b_l) \Delta_l^k$$

is the **Abelian Group of k-dimensional simplicial integer-valued chains** of a simplicial complex K, denoted by $C_k(\mathcal{K})$.

In the case of the study, in which the number of simplexes is finite, the previous groups have a finite number of generators, so they are finitely generated Abelian groups. Then it is possible to associate to each simplicial complex the corresponding set of Abelian groups $C_0(\mathcal{K}), \dots, C_n(\mathcal{K})$.

We introduce a linear operator that works between them: the **boundary operator**. Given an element Δ^k in K, its algebraic boundary is a $(k-1)$-dimensional chain in $C_{k-1}(\mathcal{K})$. This boundary inherits an orientation from one of the simplex. We assume that this orientation is given by $(-1)^i$ where i is the vertex that is absent in the particular face.

Definition 4. The **boundary** $\partial\Delta^k$ of an oriented simplex Δ^k is the sum of all its (k-1)-dimensional faces taken with induced orientation.

In the algebraic language, the boundary turns out to be

$$\partial\Delta^k = \sum_{i=0}^{k}(-1)^k\Delta_i^{k-1}.$$

In the general setting, we can extend the operator by linearity to a general element of $C_k(\mathcal{K})$,

Definition 5. The **boundary** of $c \in C_k(\mathcal{K})$ is an element ∂c in $C_{k-1}(\mathcal{K})$ given the explicit formula $\partial c = a_1\partial\Delta_1^k + \cdots + a_r\partial\Delta_r^k$ if $c = a_1\Delta_1^k + \cdots + a_r\Delta_r^k$.

This operator has some properties:

1. The operator ∂ is linear
2. The square of the boundary operator ∂ is identically zero (the boundary of a simplex has no boundary)
3. If $\partial(ac) = 0$ and $a \neq 0$ then $\partial c = 0$

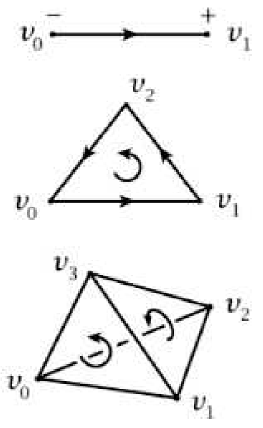
To build the *homology groups*, we need to consider the kernel and the image of such an operator. First, we introduce the following,

Definition 6. A chain z is a **cycle** if its boundary is equal to zero, i.e. $\partial z = 0$. A chain b is called a **boundary** if it is the boundary of a chain of dimension greater than 1 unit, i.e. $b = \partial h$.

Definition 7. The **set of all k-cycles** is an Abelian group, denoted as $Z_k(\mathcal{K})$ and a subgroup of $C_k(\mathcal{K})$. The **set of boundaries** forms an Abelian group, denoted by $B_k(\mathcal{K})$ and it is a subgroup of $Z_k(\mathcal{K})$.

Thanks to all these notions and notations, we introduce the main topic of this section: the **Homology**.

Definition 8. Let $z \in Z_k(\mathcal{K})$ be a k-dimensional cycle, we say that z is homologic to zero, $z \sim 0$ if $z = \partial h$ with h a certain (k+1)-dimensional chain. More generally, given z_1, z_2 k-cycles, $z_1 \sim z_2$, $z_1 - z_2 \sim 0$ if $z_1 = z_2 + \partial h$. The relation of equivalence $\sim$ is called **Homology**.



$$\partial[v_0, v_1] = [v_1] - [v_0]$$

$$\partial[v_0, v_1, v_2] = [v_1, v_2] - [v_0, v_2] + [v_0, v_1]$$

$$\partial[v_0, v_1, v_2, v_3] = [v_1, v_2, v_3] - [v_0, v_2, v_3] + [v_0, v_1, v_3] - [v_0, v_1, v_2]$$

Figure 3.4: Boundary of simplices

Since $B_k(\mathcal{K}) \subset Z_k(\mathcal{K})$ the quotient space is well defined, and so it is possible to introduce,

Definition 9. Given a simplicial complex $\mathcal{K}$, the **k-dimensional integer-valued simplicial homology group** of $\mathcal{K}$ is defined as

$$H_k(\mathcal{K}) := Z_k(\mathcal{K})/B_k(\mathcal{K}).$$

It is an Abelian Group.

For instance, $H_0(\mathcal{K})$ collects connected components (0-dim. holes), $H_0(\mathcal{K})$ cycles (1-dim. holes), $H_2(\mathcal{K})$ cavities/voids (2-dim. holes) and so on.

Some interesting and meaningful properties of $H_k(\mathcal{K})$ are collected in (see [56]),

Corollary 3.1.1. *If $\mathcal{K}$ is a simplicial complex of dimension n, then:*

- *$H_k(\mathcal{K})$ is finitely generated for every $k \geqslant 0$*
- *$H_k(\mathcal{K}) = 0$ for all $k > n$*
- *$H_n(\mathcal{K})$ is a free Abelian group*

Thanks to the fact that such groups are finitely generated, the following theorem holds

thm 3.1.2. *Let G be a finitely generated abelian group.*

- *$G = F \oplus T$, where F is a free Abelian group of finite rank $r \geqslant 0$ and T is finite*

- *T is the direct sum of cyclic groups, $T = S_1 \oplus \cdots \oplus S_k$, with order $S_i = b_i$, and with $b_1|b_2|\dots|b_k$, i.e. $b_1|b_2$ means b_1 divides b_2. The number b_i is called the torsion coefficient of G.*
- *rank(F) and the torsion coefficients are invariants of G and two finitely generated abelian groups are isomorphic if and only if they have the same rank and the same torsion coefficients.*

Remark 1: If the coefficients are taken from a field $\mathbb{K}$, then $H_k(\mathcal{K})$ is a vector space $\forall k$.

Remark 2: The element of $H_k(\mathcal{K})$, as well known, is the class of equivalence, [z]. In such a group can be present elements such as $mz \sim 0$ but $[z] \neq [0]$. The presence of these finite-order elements avoids the homology group $H_k(\mathcal{K})$ to be a free Abelian group. Thus thanks to the previous theorem, $H_k(\mathcal{K})$ can be written as

$$H_k(\mathcal{K}) = \underbrace{\mathbb{Z} \oplus \cdots \oplus \mathbb{Z}}_{\beta_k} \oplus \mathbb{Z}_{p_1} \oplus \cdots \oplus \mathbb{Z}_{p_l}$$

where β_k is equal to the copies of $\mathbb{Z}$ in the decomposition and $\mathbb{Z}_{p_i}$ are finite cyclic Abelian groups of orders p_i as in the theorem.

Definition 10. The number β_k is the k-dimensional **Betti number** of a simplicial complex $\mathcal{K}$.

Given a simplicial complex K with corresponding Abelian Homology Groups $H_0(\mathcal{K}), \dots, H_n(\mathcal{K})$ we have

Definition 11. The **chain complex** of a given simplicial complex $\mathcal{K}$, is the sequence

$$0 \to C_n(\mathcal{K}) \xrightarrow{\partial_n} C_{n-1}(\mathcal{K}) \xrightarrow{\partial_{n-1}} \dots \xrightarrow{\partial_2} C_1(\mathcal{K}) \xrightarrow{\partial_1} C_0(\mathcal{K}) \xrightarrow{\partial_0} 0$$

With this background and settings, we have to keep in mind that to use all the theories explained above we need a simplicial complex to work with. Indeed, we do not have already this structure but only some points of a domain. In the following section, it will be more clear how to create the simplicial complex structure and how to use it for our purposes.

In the context of simplicial complex another foundamental concept is the notion of Link, that can be seen as a generalization of the neighborhood. The link of a vertex encodes information about the local structure of the complex at the vertex. For this reason, it will use in the definition of a variant of KNN method in the next chapter. More in detail,

Definition 12. Let $\mathcal{K}$ be a simplicial complex, the **Link** of a $\sigma \in \mathcal{K}$ is defined by

$$Lk_{\mathcal{K}}(\sigma) = \{\tau \in \mathcal{K} | \tau \cap \sigma = \emptyset, \tau \cup \sigma \in \mathcal{K}\}.$$

Finally,

Definition 13. The **Star** of a simplex $\sigma \in \mathcal{K}$, is

$$St_{\mathcal{K}}(\sigma) = \{\tau \in \mathcal{K} | \sigma \leq \tau\}.$$

3.2 Persistent Homology (PH)

In the context of Data analysis, user usually has only a dataset $\mathcal{X} = \{\mathbf{x}_k\}_{k=1,\dots,m}$ that comes from a manifold $\mathcal{M}$ or a topological space (X, τ), or simply X, and no simplicial complex structure at hand. It is indeed in this case that PH [32] helps to compute topological invariants of finite structures. The main objective is to compute homological information of the topological space X using only available data $\mathcal{X}$.

We start considering the spaces

$$\mathbb{X}_\epsilon = \bigcup_{i=1}^{m} B(x_i, \epsilon)$$

where $B(x_i, \epsilon)$ denotes the ball centered at x_i with radius $\epsilon > 0$. If ϵ is big enough, $\mathbb{X}_{\bar{\epsilon}}$ cover completely the space X and it could suggest that $\mathbb{X}_{\bar{\epsilon}}$ could inherit also topological properties of X in addition to geometric ones. Unfortunately, this kind of approach has revealed some drawbacks and ends up being unstable. Here comes the motivation under the introduction of PH. From [14] a first example of the simplicial complex which can be constructed from X is the *Čech Complex*.

First we recall,

Definition 14. Let X be a topological space, a family of subsets of $X, \mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ with $U_\alpha \subset X \ \forall \alpha \in A$ is a **covering** of X if

$$\bigcup_{\alpha \in A} U_\alpha \supseteq X.$$

Definition 15. Given $\mathcal{U}$ a covering of X, the **nerve** of $\mathcal{U}$, denoted bu $N\mathcal{U}$, is the abstract simplicial complex with vertex set A, and where a family $\{\alpha_0, \dots, \alpha_k\}$ spans a k-simplex if and only if

$$U_{\alpha_0} \cap \cdots \cap U_{\alpha_k} \neq \emptyset.$$

$N\mathcal{U}$ is also known as Nerve complex of $\mathcal{U}$.

This construction has revealed extremely useful in theory because the Nerve Theorem (see [14]) affirms that under some hypotheses $N\mathcal{U}$ is "equal" to the underlying space X, from a topological point of view. More precisely, it outlines that $N\mathcal{U}$ is homotopy equivalent to X.

To translate the previous idea to our discrete dataset $\mathcal{X}$, assuming that $\mathcal{X}$ comes from a metric space, we can consider the aforementioned $\mathbb{X}_\epsilon$, with $\epsilon > 0$.

Definition 16. The nerve of the set of ϵ-balls centered at points of $\mathcal{X}$ is the **Čech Complex**, denoted by $\check{\mathrm{C}}(\mathcal{X}, \epsilon)$.

Thanks again to the Nerve Theorem, it is possible to prove that under some hypothesis $\check{\mathrm{C}}(\mathcal{X}, \epsilon)$ is homotopy equivalent to X. Even if it works well theoretically, problems arise when one tries to compute it. The construction of this simplicial complex for some $\epsilon > 0$ needs, for any subset of vertices, to solve a system of inequalities in order to find out if the intersection between ϵ-balls is empty or not. This is the reason why, in the applications, data analysts usually compute another kind of structure, the *Vietoris-Rips complex.*

Definition 17. Let $\mathcal{X}$ be taken from a metric space (X, d). The **Vietoris-Rips complex** for $\mathcal{X}$, associated to the parameter ϵ, denoted by $VR(\mathcal{X}, \epsilon)$, is the simplicial complex whose vertex set is $\mathcal{X}$ and $\{\mathbf{x}_0, \ldots, \mathbf{x}_k\}$ spans a k-simplex if and only if $d(\mathbf{x}_i, \mathbf{x}_j) \leqslant 2\epsilon$ for all $0 \leqslant i, j \leqslant k$. It will be denoted as $VR(\mathcal{X}, \epsilon)$.

The following result makes sure the homotopy equivalence is preserved. In fact,

Proposition 3.2.1. *We have the following inclusions,*

$$\check{C}(\mathcal{X}, \epsilon) \subset VR(\mathcal{X}, 2\epsilon) \subset \check{C}(\mathcal{X}, 2\epsilon).$$

In literature, there are also other possible definitions of abstract simplicial complexes as, for example, [15], Alpha Complex, Voronoi complex, Strong and Weak Witness Complex, and so on.

In applications, the most used is the Vietoris-Rips one because it needs only to compute the distance matrix between vertices and then filtered distances using ϵ. From now on, we consider only this simplicial complex but the results that we will present can be applied also to other simplicial complexes.

As previously mentioned, PH analyzes not only one simplicial complex but a nested sequence of them at different scales and, following the evolution of such structure, it notes down features that gradually emerge. This brief introduction does not claim to be exhaustive therefore we invite interested readers to refer, for instance, to the works [29], [56], [24], [25], [14] and [15].

We now introduce the key concept of a *filtration* of a simplicial complex.

Definition 18. A **filtration** of a simplicial complex $\mathcal{K}$ is a nested family of subcomplexes $(\mathcal{K}_t)_{t\in T}$, where T is a totally order set, such that $\forall t_1, t_2 \in T$ with $t_1 < t_2$ then $\mathcal{K}_{t_1} \subset \mathcal{K}_{t_2}$ and $\mathcal{K} = \bigcup_{t\in T} \mathcal{K}_t$.

In application most of the time it is $T \subset \mathbb{R}$.

Some examples of filtrations are:

- The previous definition can be extended into the context of topological space X. If $f : X \to \mathbb{R}$, then the family $(\mathcal{K}_t)_{t\in T}$ with $T \subset \mathbb{R}$ defines the so called **sublevel set filtration**.
- Given a subset $\mathcal{X}$ of a compact metric space, the family of Vietoris-Rips complexes $(VR(\mathcal{X}, \epsilon))_{\epsilon\in\mathbb{R}}$ and the Čech complexes $(\check{C}(\mathcal{X}, \epsilon))_{\epsilon\in\mathbb{R}}$ are filtrations.
- Given a simplicial complex $\mathcal{K}$ with vertex set V and given $f : V \to \mathbb{R}$ then f can be extended to all simplices of $\mathcal{K}$, if $\sigma \in \mathcal{K}$ is $\sigma = [v_0, \dots, v_k]$, by

$$f([v_0, \dots, v_k]) = \max_{j=0,\dots,k} f(v_i)$$

  Then the family of subcomplexes $\mathcal{K}_{t\in T} = \{\sigma \in \mathcal{K} | f(\sigma) \leq t\}$ defines a filtration call the **sublevel sets filtration** of f or **lower star filtration** of f.

In the below section of Numerical tests, this function f will be, for example, the gray-scale values at each pixel for images, the heat kernel signature for datasets as SHREC14 [52], the weight function of edges for graphs, and so on. We now recall the main theoretical results related to point cloud data but all of them can be easily applied to other contexts.

In application the most used filtration is the Vietoris-Rips one, therefore in the following, we focus mainly on this kind of filtration.

So letting $0 < \epsilon_1 < \cdots < \epsilon_l$ be an increasing sequence of real numbers, we obtain the **filtration**

$$\emptyset = \mathcal{K}_0 \subseteq \mathcal{K}_1 \subseteq \mathcal{K}_2 \subseteq \cdots \subseteq \mathcal{K}_l$$

with $\mathcal{K}_i = VR(\mathcal{X}, \epsilon_i)$.

On one side for each simplicial complex $\mathcal{K}_i$ of such a filtration we have the related chain complexes, on the other hand, the inclusion map allows us to move along the filtration. A simple complex is defined as a family of chain complexes $\{C_*^i\}_{i\geq 0}$ over a commutative ring R, together with the maps:

$$f^i : C_*^i \to C_*^{i+1}.$$

So we end up with a structure similar to the diagram here below.

$$\begin{tikzcd}
\vdots \arrow[d, "\partial_3"] & \vdots \arrow[d, "\partial_3"] & \vdots \arrow[d, "\partial_3"] & \\
C_2^0 \arrow[r, "f^0"] \arrow[d, "\partial_2"] & C_2^1 \arrow[r, "f^1"] \arrow[d, "\partial_2"] & C_2^2 \arrow[r, "f^2"] \arrow[d, "\partial_2"] & \cdots \\
C_1^0 \arrow[r, "f^0"] \arrow[d, "\partial_1"] & C_1^1 \arrow[r, "f^1"] \arrow[d, "\partial_1"] & C_1^2 \arrow[r, "f^2"] \arrow[d, "\partial_1"] & \cdots \\
C_0^0 \arrow[r, "f^0"] \arrow[d, "\partial_0"] & C_0^1 \arrow[r, "f^1"] \arrow[d, "\partial_0"] & C_0^2 \arrow[r, "f^2"] \arrow[d, "\partial_0"] & \cdots \\
0 \arrow[r, "f^0"] & 0 \arrow[r, "f^1"] & 0 \arrow[r, "f^2"] & \cdots
\end{tikzcd}$$

Definition 19. Given a filtration $(\mathcal{K}_t)_{t\in T}$ and fix $n \in \mathbb{N}$. For any $s < t$, the The **(s,t)-persistent homology group** of $\mathcal{X}$ is defined as

$$H_k^{s,t}(\mathcal{X}) = Z_k(\mathcal{K}_s)/(Z_k(\mathcal{K}_s) \cap B_k(\mathcal{K}_t)).$$

This group contains all stable homology classes in the interval i to $i+p$: they are born before the time/index i and are still alive after p steps. The persistent homology classes alive for large values of p are stable topological features of $\mathcal{S}$ (see [4]). Along the filtration, the topological information appears and disappears, thus it means that they may be represented with a couple of indexes. If p is such a feature, it must be born in some $\mathcal{K}_i$ and die in $\mathcal{K}_j$ so it can be described as (i, j), $i < j$. We underline here that j can be equal to $+\infty$, since some features can be alive up to the end of the filtration. Hence, all such topological invariants live in the extended positive plane, that here is denoted by $\mathbb{R}^2_+ = \mathbb{R}_{\geq 0} \times \{\mathbb{R}_{\geq 0} \cup \{+\infty\}\}$. Another interesting aspect to highlight is that some features can appear more than once and accordingly such collection of points are called multisets.

Definition 20. Let $\mathbb{K}$ be a given field. A **persistence module** over $T \subset \mathbb{R}$ is a family $\mathbb{V} = (V_t)_{t \in T}$ of $\mathbb{K}$-vector space endowed with linear map $\phi^{r,s} : V_r \to V_s$ such that:

1. $\phi^{t,t} = id$, $\forall t \in T$
2. $\phi^{s,t} \circ \phi^{r,s} = \phi^{r,t}$, $\forall r \leq s \leq t \in T$

$$\begin{tikzcd}
V_r \arrow[rr, "\phi^{r,t}"] \arrow[dr, "\phi^{r,s}"'] & & V_t \\
& V_s \arrow[ur, "\phi^{s,t}"'] &
\end{tikzcd}$$

These conditions are called **functoriality conditions**.

In particular, let $(\mathcal{K}_t)_{t \in T}$ be a filtration of a simplicial complex or a topological space. Given an integer $k \geq 0$ and considering the homology groups $H_k(\mathcal{K}_t)$ we obtain a family of vector spaces V_t, and the inclusions $\iota^{r,s} : \mathcal{K}_r \hookrightarrow \mathcal{K}_s$ $\forall r \leq s$ induce linear maps $\iota^{r,s}_* : H_k(\mathcal{K}_r) \to H_k(\mathcal{K}_s)$ at the homology level. Furthermore, these maps satisfy condition 2 of the definition of the persistence module, since it can be proved that $\iota^{r,t}_* = (\iota^{s,t} \circ \iota^{r,s})_* = (\iota^{s,t}_*) \circ (\iota^{r,s}_*)$ $\forall r \leq s \leq t$.

Under mild assumptions, a persistence module decomposes uniquely into elementary pieces, called *interval modules*.

Definition 21. An interval module $\mathbb{I}$ over $I \subset T$ is a persistence module $\mathbb{V}$ defined by:

1. $V_t = \mathbb{K}$ if $t \in I$, $V_t = \{0\}$ otherwise

2. $\forall r, s \in T$, $\phi^{r,s} = id_{\mathbb{K}}$ if $r, s \in T$, $\phi^{r,s} = 0$ otherwise.

   Notation for $I = [b, d]$:

$$\mathbb{I}_{[b,d]} := \underbrace{\{0\} \xrightarrow{0} \dots \xrightarrow{0} \{0\}}_{t<b} \xrightarrow{0} \underbrace{\mathbb{K} \xrightarrow{id_{\mathbb{K}}} \dots \xrightarrow{id_{\mathbb{K}}} \mathbb{K}}_{b \le t \le d} \xrightarrow{0} \underbrace{\{0\} \xrightarrow{0} \dots \xrightarrow{0} \{0\}}_{t>d}$$

thm 3.2.2. *Let $\mathbb{V}$ a persistence module indexed by $T \subset \mathbb{R}$. If T is a finite set or if all the vector spaces V_r are finite-dimensional, then $\mathbb{V}$ is decomposable as a direct sum of interval modules as*

$$\mathbb{V} \simeq \bigoplus \mathbb{I}_{[b_j, d_j]}.$$

Furthermore, when it exists, the decomposition is unique (up to isomorphism and ordering of terms).

It can be proved that both previous conditions are satisfied for the persistent homology of filtrations of finite simplicial complexes, as in the case of Vietoris-Rips and Čech complexes.

Definition 22. The **Persistent Barcode (PB)** of a decomposable module $\mathbb{V}$ is the collection of intervals $[b_j, d_j]$ of the decomposition, counted with their multiplicities.

Analogously, topological information can be grouped into the following

Definition 23. A **Persistence Diagram (PD)** $D_r(\mathcal{X}, \boldsymbol{\varepsilon})$ related to the filtration $\emptyset \subset \mathcal{K}_1 \subset \mathcal{K}_2 \subset \dots \subset \mathcal{K}_l$ with $\boldsymbol{\varepsilon} := (\epsilon_1, \dots, \epsilon_l)$ is a multiset of points defined as

$$D_r(\mathcal{X}, \boldsymbol{\varepsilon}) := \{(b, d) | (b, d) \in P_r(\mathcal{X}, \boldsymbol{\varepsilon})\} \cup \Delta$$

where $P_r(\mathcal{X}, \boldsymbol{\varepsilon})$ denotes the set of r-dimensional birth-death couples that came out along the filtration, each (b, d) is considered with its multiplicity, while points of $\Delta = \{(x, x) | x \geq 0\}$ with infinite multiplicity. One may consider all $P_r(\mathcal{X}, \boldsymbol{\varepsilon})$ for every r together, obtaining the total PD denoted here by $D(\mathcal{X}, \boldsymbol{\varepsilon})$, that we will usually consider in the following sections.

Each point $(b, d) \in D_r(\mathcal{X}, \boldsymbol{\varepsilon})$ is called **generator** of the persistent homology, and represents a topological property which appears at K_b and disappears at K_d. The difference $d - b$ is called **persistence** of the generator, represents its lifespan and shows the robustness of the topological property.

In the previous definition, the set Δ is added to finding out proper bijections between sets, that without Δ could not have the same number of points. It makes it possible to compute the proper distance between PDs.

We give here an example that clears up how these topological features appear and how they indeed represent the intrinsic property of the starting surface. We want to obtain the same results explained at the beginning of the chapter. Starting from samples of a sphere, we compute persistent features and represent them using the Persistence Diagram and Persistent Barcode.

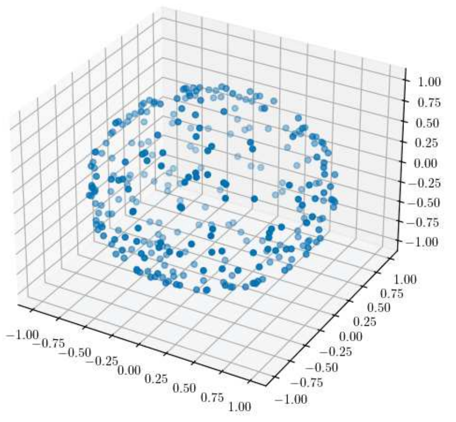
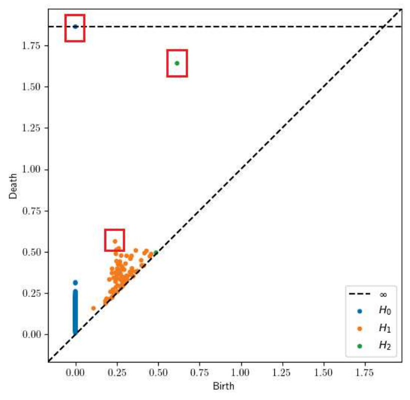



Figure 3.5: Points sampled from the sphere (left) and related PD (right)

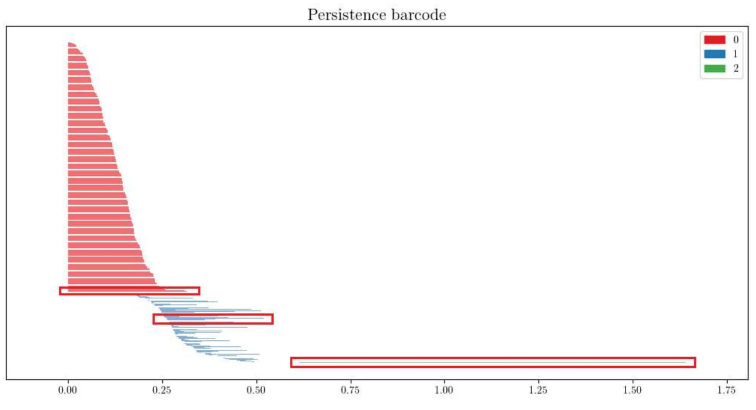


Figure 3.6: PB of samples from the sphere

Figure 3.5 (left) shows samples from the sphere. Figure 3.5 (right) is an example of the total PD collection features of dimension 0 (in blue), of dimension 1 (in orange), and of dimension 2 (in green). Points close to the diagonal represent features with a short lifetime, and so usually they are concerned with noise; instead, features far away are indeed relevant and meaningful, and, based on applications, one can decide to consider both or only the most interesting ones. At the top of the figure, there is a dashed line that indicates infinity and allows us to plot also couples as $(i, +\infty)$. In red, we highlight the most important features: 1 connected component, 1 cycle, and 1 cavity.

In Figure 3.6, we report the corresponding PB. Each line corresponds to a persistent feature, the counterpart of points in PD. The longer a line is the more important the feature is. As previously, we marked in red the main features, according to those cited at the beginning of the chapter.

3.2.1 Stability

A key property of PDs is stability under perturbation of the data. First, we recall two famous distances for sets,

Definition 24. For two nonempty sets $\mathcal{X}, \mathcal{Y} \subset \mathbb{R}^d$ with the same cardinality, the **Haussdorff distance** is

$$d_H(\mathcal{X}, \mathcal{Y}) := \max\{\sup_{x\in\mathcal{X}} \inf_{y\in\mathcal{Y}} \|x-y\|_\infty, \sup_{y\in\mathcal{Y}} \inf_{x\in\mathcal{X}} \|y-x\|_\infty\}.$$

We recall the $p - Wasserstein$ distance, $p > 0$,

$$d_{W,p}(\mathcal{X}, \mathcal{Y}) = \inf_{\gamma} \sum_{x\in\mathcal{X}} \|x-\gamma(x)\|_\infty^p$$

where $\Gamma = \{\gamma : \mathcal{X} \to \mathcal{Y} | \gamma \text{ bijection}\}$. In particular, taking $p \to +\infty$, the **bottleneck distance** can be obtained as

$$d_B(\mathcal{X}, \mathcal{Y}) := d_{W,\infty}(\mathcal{X}, \mathcal{Y}) = \inf_{\gamma\in\Gamma} \sup_{x\in\mathcal{X}} \|x-\gamma(x)\|_\infty \tag{3.1}$$

where we use

$$\|v-w\|_\infty = \max\{|v_1-w_1|, |v_2-w_2|\}, \quad \text{for } v=(v_1,v_2), w=(w_1,w_2) \in \mathbb{R}^2$$

We try to explain better how to compute the bottleneck distance. We have to take all possible ways to move points from $\mathcal{X}$ to $\mathcal{Y}$ in a bijective manner and then one

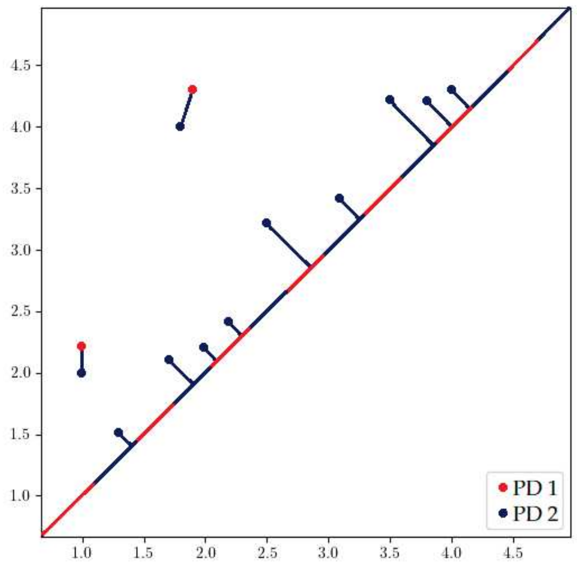


Figure 3.7: Example of bottleneck distance between two PDs in red and blue

can compute properly the distance. Figure 3.7 shows two different PDs overlapped, that consist of Δ joined with 2 points in red and 11 points in blue respectively. First, in order to apply the definition (3.1), we need two sets with the same cardinality. For this aim, it is necessary to add points of Δ, more precisely the orthogonal projection onto the diagonal of the 9 blue points closer to it, to reach 11. Lines between points and Δ represent the bijection that realizes the best matching between points in definition (3.1).

Proposition 3.2.3. *Let $\mathcal{X}$ and $\mathcal{Y}$ be finite subset in a metric space (M, d_M). Then the Persistence Diagrams $D(\mathcal{X}, \varepsilon)$, $D(\mathcal{Y}, \varepsilon)$ satisfy*

$$d_B(D(\mathcal{X}, \varepsilon), D(\mathcal{Y}, \varepsilon)) \leqslant d_H(\mathcal{X}, \mathcal{Y}).$$

For any further details see for example [56].

3.2.2 Python Libraries for TDA

Due to the robustness of the theory and to the interesting and promising results in practice, TDA and mainly Persistent Homology have received a lot of attention in the last twenty years. This is evident also when looking at how many different libraries have been developed to allow user all over the world to compute and manage such a topological machine. In literature can be found libraries with different languages of programming as Python, R, Java, and so on but here we recall only the most famous ones related to the Python world, which is also the most popular. Thus the main famous Python libraries for TDA are:

- **Ripser** [64]: it is a lean persistent homology package for Python. Building on the blazing fast C++ Ripser package as the core computational engine, mainly it can visualize persistence diagrams and compute lower star filtrations on images,

  `https://ripser.scikit-tda.org/en/latest/`

- **Gudhi** [70]: is a generic open source C++ library, with a Python interface, for Topological Data Analysis (TDA) and Higher Dimensional Geometry Understanding. The library offers state-of-the-art data structures and algorithms to construct simplicial complexes, compute persistent homology, show persistence diagrams and persistent barcodes, prune a filtration.

  `https://gudhi.inria.fr/`

- **Giotto-tda** [71]: it is a high-performance topological machine learning toolbox in Python built on top of scikit-learn and is distributed under the GNU AGPLv3 license. It allows us to apply the theory of PH to a lot of different kind of data, such as points cloud data, images, graphs, and series as well as Persistence Images, Betti curve,s and Persistence Landscapes.

- **Dionysus** [75]: it is a computational topology package focused on persistent homology. It is written in C++, with Python bindings. It may compute filtration, Persistence Homology, and distances among PD and plot the results into Persistence Diagrams.

- **DIPHA** [76]: This C++ software package computes persistent homology. Besides supporting parallel execution on a single machine, DIPHA may also be run on a cluster of several machines using MPI.

Chapter 4

Classification with Persistent Homology

Nowadays, *classification* is a relevant task in life, where a big amount of data can be stored, accessible, and used to make all kinds of decisions in many areas such as medicine, economics, applied science, and more.

A very common example of such a problem is detecting SPAM e-mails. For instance, one wants to provide an algorithm able to filter out if an incoming e-mail is SPAM or not. During the so called *Training Phase*, the algorithm analyzes a group of e-mails labeled as SPAMs and a group of regular ones in order to find out patterns and features that can make it able to distinguish them. This set of examples is known as *Training Set*. After that, the algorithm can predict, hopefully in a satisfactory manner, if a new incoming e-mail is SPAM or not. This is the case of the supervised classification problem.

This task takes its origin some time ago since, for example, the k-nearest neighbors algorithm, a possible method to use, was developed in 1951. During the years, a lot of different methods and variants have been developed to solve the problem trying to face issues and drawbacks that the baseline methods could present. The most famous baseline methods are: K-th nearest neighbors (KNN), Support Vector Machine (SVM), Decisional Tree (DT), and Random Forest, only to name a few. In what follows, we focus mainly on KNN and SVM.

From a mathematical point of view, let $\Omega \subset \mathbb{R}^d$, $\mathcal{X}_l = \{\mathbf{x}_1, ..., \mathbf{x}_m\}, \mathcal{X}_u \subset \Omega$ be set of labeled points and unlabelled ones respectively. Let $Y_l = \{y_1, \dots, y_m\}$ be the set of corresponding labels of points in $\mathcal{X}_l$ where $y_i \in L = \{l_1, \dots, l_s\}$, the set of labels or classes. We denote with $\mathbf{l}_j$ the vector e_j of the canonical base in $\mathbb{R}^s$. So we introduce $\mathbf{L} = \langle \mathbf{l}_1, \dots, \mathbf{l}_s \rangle$, or better the vector space isomorphic to $\mathbb{R}^s$. The set of couples $\{(\mathbf{x}_i, y_i)\}_{1\leq i\leq m}$ is the Training Set instead $\mathcal{X}_u$ can be also called Test Set and we denote their union as $\mathcal{X}_{lu} = \mathcal{X}_l \cup \mathcal{X}_u$. If $L = \{-1, 1\}$, it is called **binary problem**.

4.1 Classification with KNN

Probably the first, simple, intuitive, and widely used method that appears in literature was Kth Nearest Neighbors (KNN). The idea behind is indeed clear:

"Similar points are closer to each other."

So to determine the belonging class of a new point, the only thing to do is to infer such a prediction analyzing its neighbors. Fixed $k \in \mathbb{N}$, the algorithm takes $\mathbf{x} \in X_u$ and considers k points in X_l that are closer to it, in a suitable and declared notion of distance, for example, Euclidian norm, sup norm, Manhattan distance and so on. Once extracted these k points, the algorithm assigns to $\mathbf{x}$ the most recurrent label among them.

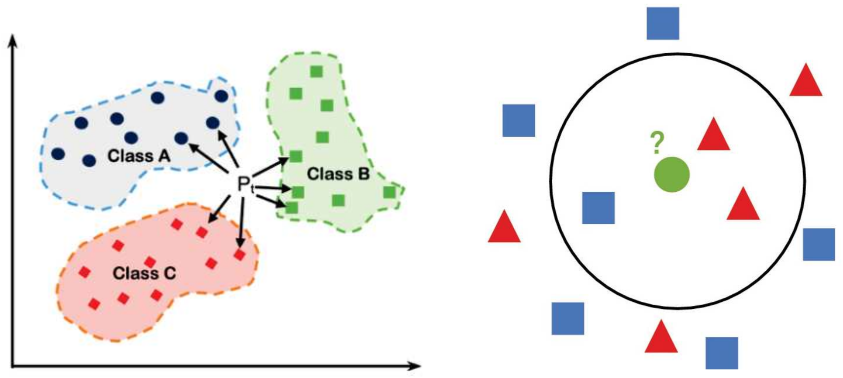


Figure 4.1: The idea of similarity of points (left) and how KNN works with ($k = 3$) (right)

4.2 Classification with KNN and PH

With the previous scheme in mind, authors in [38] have developed a new technique that infers labels exploiting the structure of data given by a simplicial complex. The authors explain a method based on **Link-based label propagation function** and the main goal is to define a proper **label function** that allows to attach the right label to an unlabeled point. From now on we will refer to it using **TDA Global**. As explained in Chapter 3, given a set of points, it is possible to build the filtration, a nested sequence of simplicial complexes, upon points. The first step of the proposed algorithm is to select a simplicial complex into this chain that can sum up, hopefully in a realistic way, the structure of points themselves, instead of going ahead until the completely connected final simplicial complex.

Before going into details about the method workflow, we discuss the choice of the selected simplicial complex $\mathcal{K}_i$ to consider. In [38] the authors have already suggested some criteria to follow in order to select a good candidate for simplicial complex. These criteria are called **selectors**. If we collect all persistent features $p = (b, d)$ into a set $P_{\mathcal{K}}$ and if we denote with $\text{pers}(p) = d - b$ the lifetime of p, some possible choices are:

- **AVG**: $p_{avg} = (\bar{b}, \bar{d})$ is the persistent feature that realizes

  $$\min_{p \in P_{\mathcal{K}}} |\text{pers}(p) - \text{avg}|$$

  where **avg** is the average of all $\text{pers}(p)$ with $p \in P_{\mathcal{K}}$.

- **HAVG**: $p_{havg} = (\bar{b}, \bar{d})$ is the persistent feature that attains

  $$\min_{p \in P_{\mathcal{K}}} |\text{pers}(p) - \text{havg}|$$

  where **havg** is the harmonic mean that, for a set of points $\{x_1, \ldots, x_n\}$, is defined as $\text{havg}(x_1, \ldots, x_n) = \frac{n}{\frac{1}{x_1} + \cdots + \frac{1}{x_n}}$

- **MAX**: $p_{max} = (\bar{b}, \bar{d})$ is the persistent feature that achieves

  $$\max_{p \in P_{\mathcal{K}}} \text{pers}(p)$$

- **MEDIAN**: $p_{med} = (\bar{b}, \bar{d})$ is the persistent feature that realizes

  $$\min_{p} |\text{pers}(p) - \text{median}|$$

  where **median** is the median of all $\text{pers}(p)$ with $p \in P_{\mathcal{K}}$.

- **RANDOM**: $p_{random} = (\bar{b}, \bar{d})$ is chosen uniformly at random among all persistent features $p \in P_{\mathcal{K}}$

After choosing one of the previous options, the selected simplicial complex turns out to be $\mathcal{K}_i = f^{-1}((-\infty, \bar{d}))$ where $p \in \{p_{max}, p_{random}, p_{med}, p_{avg}, p_{havg}\}$.

Then the authors in [38] introduce the following,

Definition 25. Let $\phi : \mathcal{X}_{lu} \to \mathbf{L}$ the **association function** defined on a vertex, or 0-dimensional simplex, $v \in \mathcal{X}_{lu}$ as $\phi(v) = \mathbf{l}_s$ for $v \in \mathcal{X}_l$, $\mathbf{0}$ $(\in \mathbb{R}^s)$ otherwise. Then its extension to any simplex in $\mathcal{K}$ is given by

$$\Phi(\sigma) = \sum_{v \in \sigma} \phi(v)$$

The previous function concretely collects recurrences of labels in a simplex $\sigma \in \mathcal{K}_i$.

Now we have to address the problem of how to assign a label to an unlabeled point. This can be done through the following **extension function**,

Definition 26. Let $\Psi : \mathcal{X}_u \to \mathbf{L}$ be a function defined on a point $\mathbf{x} \in \mathcal{X}_u$ by

$$\Psi(\mathbf{x}) = \sum_{\sigma \in Lk_{\mathcal{K}_i}(\{\mathbf{x}\})} w(\mathbf{x}, \sigma)\Phi(\sigma) = \sum_{\sigma \in St_{\mathcal{K}_i}(\{\mathbf{x}\})} w(\mathbf{x}, \sigma \setminus \{\mathbf{x}\})\Phi(\sigma \setminus \{\mathbf{x}\}) = \sum_{j=1}^{s} a_j \mathbf{l}_j$$

with proper definition of weight function w, where Lk denotes the Link and St the Star. For all details, see Chapter 3.

As in the context of KNN, the label of $\mathbf{x} \in \mathcal{X}_u$ is directly influenced by those of its neighbors. So, the present method can be considered as the generalization of the KNN idea to the structure of simplicial complexes, where it is well known how the concept of neighborhood is replaced by the Link.

Finally, the l_j corresponding to the highest a_j in the previous sum becomes the label of the point $\mathbf{x}$. More precisely,

Definition 27. Let $\tilde{a}$ the maximum in $\{a_1, \dots, a_s\}$ and $\tilde{A} = \{j | a_j = \tilde{a}\}$, the **labeling function** Υ is

$$\Upsilon(\mathbf{x}) = l_k, \text{ if } |\tilde{A}| = 1$$

otherwise

$$\Upsilon(x) = l_j \text{ with } j \text{ chosen uniformly at random in } \tilde{A}.$$

To run the algorithm is essential to define the weight function w. It is reasonable that points closer to the point $\mathbf{x} \in \mathcal{X}_u$ influence more the prediction of its label than the farther ones. But points closer means closer based on a distance and in a filtration this is equivalent to saying that they live in some simplices that are born early in it. For this reason, in its final version, w is given more or less by the inverse of the index of the simplicial complex in filtration, where that simplex appears for the first time. This concept is formalized as follows. If $f^{-1}((-\infty, \bar{\epsilon})) = \mathcal{K}_i$, we consider $\hat{f} : \mathcal{K}_i \to [0, \bar{\epsilon}]$ such that for each $\sigma \in \mathcal{K}_i$, it assigns the values $\hat{\epsilon}$ if $\sigma \in f^{-1}((-\infty, \hat{\epsilon}))$ for the first time and by definition of filtration, $\hat{\epsilon} \leq \bar{\epsilon}$.

As suggested in [38], in this way we consider the weight function w

$$w(\mathbf{x}, \sigma) = \frac{\frac{1}{f(\sigma \cup \{\mathbf{x}\})^2}}{\sum_{\mu \in St_{\mathcal{K}_i}(\{\mathbf{x}\})} \frac{1}{f(\mu)^2}}$$

After some calculations, we get the simplified expression for Ψ:

$$\Psi(\mathbf{x}) = \sum_{\sigma \in Lk_{\mathcal{K}_i}(\{\mathbf{x}\})} \frac{\Phi(\sigma)}{f(\sigma \cup \{\mathbf{x}\})^2} = \sum_{\sigma \in St_{\mathcal{K}}(\{\mathbf{x}\})} \frac{\Phi(\sigma \setminus \{\mathbf{x}\})}{f(\mu)^2}.$$

Since the method is indeed innovative and interesting from a theoretical point of view, in application we have found some drawbacks that have brought us to create its "local version", named **TDA Local**.

4.2.1 TDA Local

In our tests, we have taken into account datasets of different dimensions, from 50 to more than 3000 points. Although datasets are not huge, TDA Global has had some problems. More precisely, the problem is related to the computation or mainly to the storage of the simplices of the filtration, once computed. Keeping in mind the construction of the Vietoris-Rips complex, it is evident how the growth of the number of simplices is exponential as ϵ increases and obviously the higher the number of points is, the higher the number of simplices of the filtration is making the memory problem remarkable. To be able to use TDA Global with all datasets, a possible solution is to reduce the number of simplices or better take only the simplices with dimensions up to a certain value `max_dim`. Even if it seems to fix the problem, on the other hand, the reduction of dimension of simplices could make the algorithm less accurate. For this reason, we propose to consider a local variant of this method, called **TDA Local**.

Inspired by the idea of KNN, to determine the label of $\mathbf{x} \in \mathcal{X}_u$ we suggest to consider only a cluster of K points centered on x and then apply TDA Global only to this small dataset, *local dataset*. Restricting the number of points reduces the number of simplices of the filtration and therefore the memory needed. In this way, it is also possible to slightly increase the value of `max_dim` allowing to use also simplices with higher dimension too with the purpose of making the prediction more accurate.

First of all, TDA Local, since it is localized, depends on a particular parameter K, remembering in a certain way the standard KNN algorithm. It could not be considered a hyperparameter properly, but it is only a feature to set. More precisely, in this context, the meaning of k is slightly different from that in the KNN context. Here, K allows to make a zoom of the dataset restricting the number of points to consider for computations; instead in KNN, k denotes the number of points that one decides to consider as significant and more influential for determining the final label. To distinguish the two concepts, here we denote the number with the capital letter K. Of course, like TDA Global, TDA Local also has a selector to set.

The construction of a TDA Local follows the following steps:

1. Set Selector and value for K
2. Take a point $\mathbf{x} \in \mathcal{X}_u$ and extract the first K-th nearest neighbors using only those from $\mathcal{X}_l$, creating a new *local dataset*. This consideration avoids to encounter situations as Star is empty, as previously mentioned
3. Compute simplicial complex and apply the TDA Global to the *local dataset*

At the moment, TDA Global and TDA Local have been developed in Python and are ready to be converted into suitable files to be integrated into preexisting software SMART Prod Active, see Chapter 2. The concrete integration is in progress, and probably in some months some tests could be done in order to evaluate the performances of these topological models.

4.2.2 Description of the software

For what concerns the Python code, we have written down our own implementation of the method. First, we have built a proper class `TDAClassifier` where we have collected all the essential methods needed to perform the classification. To be created, an instance of `TDAClassifier` needs information about the dataset to classify and `label_complex`, that outlines the kind of simplicial complex to build. We consider only the most common one, the Vietoris-Rips complex, but any other construction can be easily added to the code, using `gudhi` library. After that, the user must choose which selector to apply among the 5 available calling the `PruneFiltration` method of the class, ending with the selected simplicial complex $\mathcal{K}_i$. In the code, we use only this library for all concerns topological aspects since there are some functionalities just available: class `RipsComplex` and its method `create_simplex_tree` to compute and store efficiently the structure of filtration, the method `prune_above_filtration` to cut off all simplices over a certain index and `get_star`, a method that allows computing the star of a specific simplex σ. Then, calling the class method `MakePredictions` Using the instance of the TDAClassifier, it is possible to apply the method to each $\mathbf{x} \in \mathcal{X}_u$.

Given $\mathbf{x} \in \mathcal{X}_u$, the code computes its star and then calls the method `compute_star_for_classification`. In the end, its output consists of two lists:

- `labelled_star`: it stores all simplices of the neighbourhood of $\mathbf{x}$
- `labelled_epsilon_val`: for each simplex of the previous star, it collects the epsilon values of the simplicial complex where such a simplex appears for the first time

Finally, the function Υ effectively makes the prediction by summing up all the information from the previous steps. Averaging on the number of splits, one ends up with the final score.

But there are some possible issues that can be addressed and understood how to manage. Three are the main problems, and we explain how to possibly overcome them:

- `simplicial_neighbourhood`: the point $\mathbf{x}$ could be a little isolated from other points in $\mathcal{X}_{lu}$. So, it can happen that after pruning, its star is empty. In that case, the function enlarges a little bit the pruning values, considering points whose distance is twice the one previously considered. This trick is usually enough to fix the problem.

Now, if the original star, Star1 is not empty, the test points have been removed from it obtaining a new star, Star2.

- `compute_filtered_star_unlabelled_points`: Star2 may be empty in the case where $\mathbf{x}$ is completely surrounded by points in $\mathcal{X}_u$. The function takes Star1 and, for each simplex, calls `compute_filtered_star` again and collects all contributions.

Finally, if the `labelled_star` is empty again, the method applies KNN with $K = 1$.

The aforementioned drawbacks have to be faced to allow the code to run but, in our application, at least in the case of optimal choice of selector, they are not so recurrent, affecting only at most 2% of unlabeled samples. This means that the code is working correctly.

Concerning the Python implementation of our method, **TDA Local**, its main core has been inherited from TDA Global. So we can apply the previous code locally, to each point $\mathbf{x}$ and don't need to develop any further implementations.

4.2.3 Numerical Tests

Inspired by the GitHub code proposed in [38] and after having analyzed it deeply, we have written down our own code, which is available on GitHub.

```
https://github.com/cinziabandiziol/TDA_classification
```

All codes have been run using Python 3.11 on a 2.5 GHz Dual-Core Intel Core i5, 32 Giga RAM. In tests, we use the implementation of SVM, KNN, and DT provided by the `Scikit` [49] library of Python.

In addition to the foundry data described in Chapter 2, we have also taken into account other datasets of different nature. Two of them have been computed numerically using the `sklearn.datasets` library.

- **MOON**: it collects data that lie on two interleaving half circles
- **CIRCLES**: it consists on data that lie on a large circle containing a smaller one in 2d

Some datasets are taken from the **UC Irvine Machine Learning Repository**, a very common repository in the Machine Learning community and easy to import in program languages such as Python and R,

- **DIABETIC RETHINOPATHY [DIAB. RET.]**: it contains features extracted from the Messidor image set to predict whether an image contains signs of diabetic retinopathy or not.
- **RICE**: it collects the morphological characteristics obtained for each grain of rice: Osmancik and Cammeo.

- **SURGERY**: dataset collects information about patients who underwent major lung resections for primary lung cancer
- **LIVER**: it contains information about ill and healthy patients related to concentrations of some chemical compound in blood

Finally, the other ones are downloaded from the Python library `sklearn.dataset`,

- **IRIS**: it consists of different attributes such as sepal length, sepal width, petal length, petal width and species to be used to distinguish among 3 varieties of Iris flowers
- **CANCER**: it collects real features related to breast cancer Wisconsin dataset
- **WINE**: the dataset contains information about chemical compositions, color, hue, and percentage of alcohol of 3 different kinds of wine

These datasets which we group here under the name of *general datasets*, represent a heterogeneous selection of data, with respect to cardinality, number of classes, number of features, and balance or imbalance nature.

In Table 4.1, a summary of the main characteristics of the aforementioned datasets is presented where column `IR` expresses the imbalance of the data. If it equals 1, the dataset is completely balanced; instead, the higher this index is, the more imbalanced the dataset is. See Appendix for details about IR.

Dataset	**N° samples**	**N° classes**	**IR**
CIRCLES	50	2	25:25
IRIS	150	3	50:50
WINE	178	3	71:48
MOON	200	2	100:100
SURGERY	470	2	400:70
CANCER	570	2	357:213
LIVER	580	2	413:167
DIAB. RET.	1080	2	540:540
RICE	3260	2	1630:1630

Table 4.1: Datasets for classification

For all experiments, we use K-Fold CV or its Stratified version (see Appendix), with different numbers of folds, depending on the dataset at hand. For each of them in class `Dataset` the variable `n_fold` is set according to the structure of the data and the total number of samples. A common value is 10. Since the topological methods have no proper hyperparameters to tune, here the subdivision provided by CV is useful to split the dataset into Training and Tests sets, allowing for the correct evaluation of the performance of the model, testing them on different subdivisions, and finally taking the average of the score obtained.

4.2.4 Comparison using different selectors

First, we are interested in analyzing how the TDA classifier's power of prediction changes as the selector varies among those cited before. From its definition, it is evident how the selector turns out to be, roughly speaking, the only hyperparameter to tune in both topological methods. Thus, we have run some tests on general datasets and also on foundry data to better explore the context.

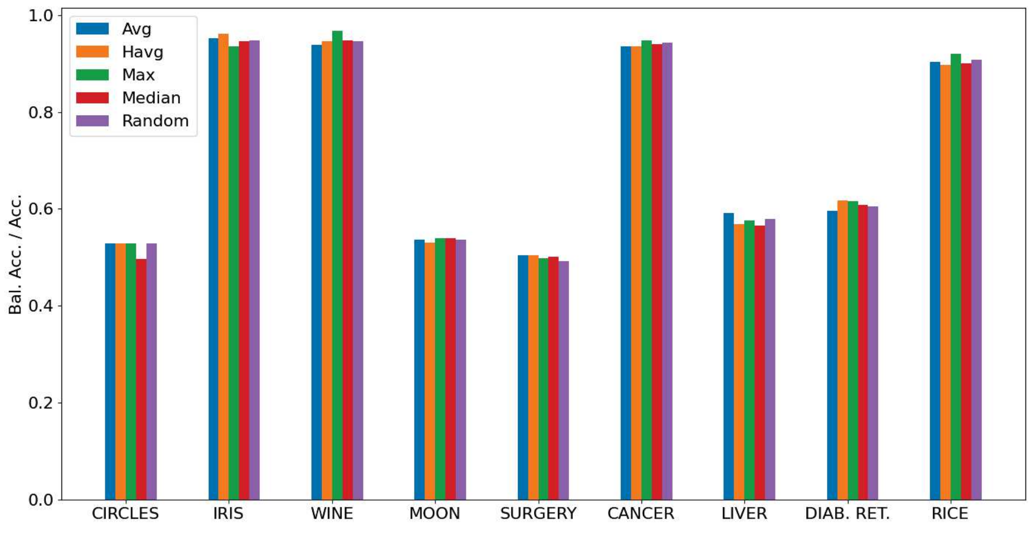


Figure 4.2: Comparisons between performances using different selectors for TDA Global

For each dataset, we have run our code for TDA Global and TDA Local, and saved the prediction performances of the TDA classifier as the selector varies. To measure the performance, we have used accuracy or balanced_accuracy in the case of balanced and imbalanced datasets, respectively. In Figure 4.2 we plot the performance of TDA Global for different datasets, as shown on the x-axis. For each of them, the

5 bars represent the values of Accuracy/Balanced_accuracy registered. We report only results related to general datasets but the same considerations can be extended also to the foundry ones. The conclusion here is that, globally, the choice of selector does not affect too much the generalization power of the topological model. For each dataset, it is evident how all bars reach more or less the same height and how they can't outline any clear winner.

On the other hand, on the TDA Local, we collect all previous results in Table 4.2, where we highlight in bold the best value for each dataset.

Dataset	AVG	HAVG	MAX	MEDIAN	RANDOM
CIRCLES	0.504	**0.529**	**0.529**	0.488	0.488
IRIS	0.936	**0.961**	0.936	0.947	0.936
WINE	**0.967**	0.946	0.946	0.953	0.951
MOON	0.513	**0.564**	0.516	0.515	0.526
SURGERY	0.479	0.518	0.497	0.489	**0.525**
CANCER	0.944	0.946	**0.952**	0.942	0.949
LIVER	0.564	**0.598**	0.565	0.571	0.579
DIAB. RET.	**0.628**	0.614	0.607	0.612	0.611
RICE	0.904	0.900	**0.915**	0.906	0.909

Table 4.2: Accuracy/Balanced_accuracy of TDA Local classifier related to different datasets (best values in **bold**)

Even if the number of datasets tested is not high, it is clear how Median and Random selectors can easily be excluded from the analysis because the related accuracy is even not the best. We can confirm that the best selector is HAVG, followed by MAX. About the foundry data, the selector's analysis brings to the same conclusion and therefore we conduct all tests considered directly HAVG selector.

4.2.5 Comparisons between different methods

The attention is now focused on another interesting analysis, the comparisons between methods. We take into account here the most three famous baseline methods, such as KNN, DT and SVM. For any further details about these methods, we suggest the reader to have a look at the Appendix. They have some hyperparameters to tune, and so for the GridSearch, we set:

- **KNN**: the hyperparameter k represents the number of neighbors to consider at each iteration of the method. So `n_neighbors` is taken among $\{1, 2, \ldots, 50\}$ and as method or `algorithm` used to compute the nearest neighbors we consider `ball_tree, kd_tree, brute`

- **DT**: for `criterion`, namely the function to measure the quality of a split, we take into account `gini, entropy, log_loss`

- **SVM**: we choose `kernel` among `linear, poly, rbf` that are equivalent to the linear kernel, the polynomial kernel of some degree, and Radial Basis Function while C is set equal to 1, as commonly done.

In terms of TDA Local, some experiments have been performed on the values of K to take into account. We know that it depends on the data at hand, but in general, for our tests, a common and good choice seems to be 100. Obviously, some deep analysis needs to be done in future work.

Dataset	**TDA Gl.**	**TDA Local**	**KKN**	**DT**	**SVM**
CIRCLES	**0.529**	**0.529**	0.383	0.396	0.296
IRIS	0.961	0.961	0.95	0.912	**0.967**
WINE	0.967	0.967	0.976	0.904	**0.977**
MOON	0.539	**0.564**	0.531	0.548	0.462
SURGERY	0.504	0.525	0.482	0.498	**0.578**
CANCER	0.948	0.952	0.959	0.918	**0.962**
LIVER	0.592	0.598	0.568	0.599	**0.701**
DIAB. RET.	0.617	0.628	0.664	0.628	**0.696**
RICE	0.921	**0.935**	0.929	0.889	0.927

Table 4.3: Comparison in terms of Accuracy/Balanced_accuracy between methods and datasets (in **bold** global best score, in red the best one among topological methods)

From results in Table 4.3 can be drawn. On general datasets, the final accuracy remains more or less the same even if we change the model used. If we focus first only on topological methods, it is evident how the performances improve using TDA Local instead of TDA Global. But if we enlarge the comparison to include all methods, the topological ones realize the best scores only for CIRCLES and MOON. Anyway, regardless of the method used, the final results are not far from each other, and therefore TDA Global and TDA Local finally represent a good representation of an interesting and possible alternative to standard baseline methods.

Variables	TDA Gl.	TDA Local	KKN	DT	SVM
Z1_1	0.381	0.410	0.406	0.390	**0.426**
Z1_2	0.678	0.702	**0.724**	0.606	0.718
Z1_3	0.543	0.624	0.668	0.51	**0.671**
Z2_2	0.587	0.540	**0.605**	0.517	0.587
Z3_1	0.601	0.637	**0.701**	0.610	0.608
Z3_3	0.466	0.486	0.557	0.55	**0.578**
Z4_1	0.647	0.645	0.632	0.612	**0.658**
Z4_3	0.674	0.701	0.725	0.652	**0.740**
PIN2	0.748	0.730	**0.776**	0.7	0.751
PIN3	0.577	0.571	**0.641**	0.577	0.614

Table 4.4: Comparison in terms of Accuracy/Balanced_accuracy between methods and datasets (in **bold** global best score, in red the best one among topological methods)

About foundry data, first, we recall that they are more difficult to treat because of their high imbalance and slightness. All results have been collected in Table 4.4. As before, comparing topological methods, we see how the winner changes as the quality variable varies. and so TDA Local doesn't emerge here as the method to prefer. However, their performances, although different, are closer to each other, making them indeed comparable. Considering also KNN, DT, and SVM, it becomes clear how topological methods seem to have some issues in the prediction phase. Of course, this consideration keeps open some possible future investigations to improve even more the models based on TDA.

4.3 Classification with SVM

Let $\Omega \subset \mathbb{R}^d$ and $\{\mathbf{x}_1, ..., \mathbf{x}_m\} \subset \mathcal{X} \subset \Omega$ be the set of input data with $d, m \in \mathbb{N}$. We have a training set, composed of the couples $(\mathbf{x}_i, y_i)$ with $i = 1, ..., m$ and $y_i \in \mathcal{Y} = \{-1, 1\}$. The **binary supervised learning task** consists of finding a function $f : \Omega \longrightarrow \mathcal{Y}$, the model, such that it can predict, in a satisfactory way, the label of an unseen $\tilde{\mathbf{x}} \in \Omega \setminus \mathcal{X}$.

As explained in the Appendix, through the kernel trick, the optimization problem to solve becomes

$$\begin{aligned}
\max_{\alpha\,\in\,\mathbb{R}^m} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i\alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \\
\text{s. to} \quad & \sum_{i=1}^{m} \alpha_i y_i = 0 \\
& 0 \leq \alpha_i \leq \frac{C}{m}\ \forall i = 1, \ldots, m
\end{aligned}$$

where the kernel represents a generalization of the inner product in $\mathbb{R}^d$. We are interested in classifying PDs and, obviously, we need suitable definitions for kernels for PDs, the so-called **Persistence Kernels (PK)**.

4.3.1 Persistence Kernels

Inspired by [23], we focused our attention on the classification with SVM through Persistence Diagrams instead of considering the original datasets. If $\mathfrak{D}$ is the set of PDs, then we consider the main kernels available. In what follows, we denote by $\mathfrak{D}$ the set of the total PDs.

Persistence Scale-Space Kernel (PSSK)

The first kernel was described in [55]. The main idea is to compute the feature map as the solution of the Heat equation. We consider $\Omega_{ad} = \{\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2 : x_2 \geqslant x_1\}$ and denote with $\delta_{\mathbf{x}}$ the Dirac delta centered at $\mathbf{x}$. For a given $D \in \mathfrak{D}$, we consider the solution $u : \Omega_{ad} \times \mathbb{R}_{\geqslant 0} \to \mathbb{R}$, $(\mathbf{x}, t) \mapsto u(\mathbf{x}, t)$ of the following PDE:

$$\begin{aligned}
\Delta_{\mathbf{x}} u = \partial_t u \quad & \text{in } \Omega_{ad} \times \mathbb{R}_{\geqslant 0} \\
u = 0 \quad & \text{on } \partial\Omega_{ad} \times \mathbb{R}_{\geqslant 0} \\
u = \sum_{\mathbf{y} \in D} \delta_{\mathbf{y}} \quad & \text{on } \Omega_{ad} \times 0.
\end{aligned}$$

The feature map $\Phi_\sigma : \mathfrak{D} \to \mathrm{L}^2(\Omega_{ad})$ at scale $\sigma > 0$ at D is defined as $\Phi_\sigma(D) = u\big|_{t=\sigma}$. This map yields the **Persistence Scale-Space Kernel** (**PSSK**) K_{PSS} on $\mathfrak{D}$ as:

$$K_{PSS}(D, E) = \langle \Phi_\sigma(D), \Phi_\sigma(E) \rangle_{\mathrm{L}^2(\Omega_{ad})}.$$

Since it is known as an explicit formula for the solution u, the kernel takes the form

$$K_{PSS}(D,E) = \frac{1}{8\pi\sigma} \sum_{\mathbf{x}\in D, \mathbf{y}\in E} \exp(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{8\sigma}) - \exp(-\frac{\|\mathbf{x}-\bar{\mathbf{y}}\|^2}{8\sigma})$$

where $\mathbf{y} = (a,b)$, $\bar{\mathbf{y}} = (b,a)$, for any $D, E \in \mathfrak{D}$.

Persistence Weighted Gaussian Kernel (PWGK)

In [39], the authors introduce a new kernel whose idea is to replace each PD with a discrete measure. Starting with a strictly positive definite kernel, as the Gaussian one $\kappa_G(\mathbf{x},\mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}}$, $\sigma > 0$ we denote the corresponding Reproducing Kernel Hilbert Space $\mathcal{H}_{\kappa_G}$.

If $\Omega \subset \mathbb{R}^d$, we denote with $M_b(\Omega)$ the space of finite signed Radon measures and

$$E_{\kappa_G} : M_b(\Omega) \to \mathcal{H}_{\kappa_G}, \mu \mapsto \int_\Omega \kappa_G(\cdot, \mathbf{x}) d\mu(\mathbf{x}).$$

For any $D \in \mathfrak{D}$, if $\mu_D^w = \sum_{\mathbf{x}\in D} w(\mathbf{x})\delta_{\mathbf{x}}$, where the weight function satisfies $w(x) > 0$ for all $\mathbf{x} \in D$ then

$$E_{\kappa_G}(\mu_D^w) = \sum_{\mathbf{x}\in D} w(\mathbf{x})\kappa_G(\cdot,\mathbf{x})$$

where

$$w(\mathbf{x}) = \arctan(C_w pers(\mathbf{x})^p)$$

and $pers(\mathbf{x}) = x_2 - x_1$.

The **Persistence Weight Gaussian Kernel** (**PWGK**) is defined as

$$K_{PWG}(D,E) = \exp\left(- \frac{1}{2\tau^2} \|E_{\kappa_G}(\mu_D^w) - E_{\kappa_G}(\mu_E^w)\|^2_{\mathcal{H}_{\kappa_G}} \right), \tau > 0$$

for any $D, E \in \mathfrak{D}$.

Sliced Wasserstein Kernel (SWK)

Another possible choice for κ has been introduced in [16].

We consider μ and ν two nonnegative measures on $\mathbb{R}$ such that $\mu(\mathbb{R}) = r = |\mu|$ and $\nu(\mathbb{R}) = r = |\nu|$, we recall that the 1-Wasserstein distance for nonnegative measures is defined as

$$\mathcal{W}(\mu,\nu) = \inf_{P\in\Pi(\mu,\nu)} \int\int_{\mathbb{R}\times\mathbb{R}} |x-y| dP(x,y)$$

where $\Pi(\mu, \nu)$ is the set of measures on $\mathbb{R}^2$ with marginals μ and ν.

Definition 28. 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 $D, E \in \mathfrak{D}$ and let $\mu_D^\theta := \sum_{\mathbf{x}\in D} \delta_{\pi_\theta(\mathbf{x})}$ and $\mu_{D\Delta}^\theta := \sum_{\mathbf{x}\in D} \delta_{\pi_\theta \circ \pi_\Delta(\mathbf{x})}$ and similarly for μ_E^θ and $\mu_{E\Delta}^\theta$ where π_Δ is the orthogonal projection onto the diagonal. Then, the **Sliced Wasserstein distance** is

$$SW(D, E) = \frac{1}{2\pi} \int_{\mathbb{S}_1} \mathcal{W}(\mu_D^\theta + \mu_{E\Delta}^\theta, \mu_E^\theta + \mu_{D\Delta}^\theta) d\theta.$$

Thus, the **Sliced Wasserstein Kernel** (**SWK**) is defined as

$$K_{SW}(D, E) := \exp\left(- \frac{SW(D, E)}{2\eta^2} \right), \ \eta > 0$$

for any $D, E \in \mathfrak{D}$.

Persistence Fisher Kernel (PFK)

In [41], the authors describe a kernel based on *Fisher Information geometry.*

A persistence diagram $D \in \mathfrak{D}$ can be considered as a discrete measure $\mu_D = \sum_{u\in D} \delta_u$, where δ_u is the Dirac's delta centered in u. Given a bandwidth $\sigma > 0$, and a set Θ, one can smooth and normalize μ_D as follows

$$\rho_D := \frac{1}{Z} \sum_{u \in D} N(\mathbf{x}; u, \sigma I)$$

where N is a Gaussian function, $Z = \int_\theta \sum_{u\in D} N(x; u, \sigma I) dx$ and I is the identity matrix. Thus, using this measure, any PD can be regarded as a point in $\mathbb{P} = \{\rho | \int \rho(\mathbf{x}) d\mathbf{x} = 1, \rho(\mathbf{x}) \geq 0\}$.

Given two element in $\rho_i, \rho_j \in \mathbb{P}$, the **Fisher Information Metric** is

$$d_{\mathbb{P}}(\rho_i, \rho_j) = \arccos\left(\int \sqrt{\rho_i(\mathbf{x})\rho_j(\mathbf{x})} d\mathbf{x} \right).$$

Inspiring by the Sliced Wasserstein Kernel construction, we have the following

Definition 29. Let D, E be two finite and bounded persistence diagrams. The Fisher information metric between D and E is

$$d_{FIM}(D, E) := d_{\mathbb{P}}(\rho_{D \cup E_\Delta}, \rho_{E \cup D_\Delta})$$

where $D_\Delta := \{\Pi_\Delta(u) | u \in D\}$, $E_\Delta := \{\Pi_\Delta(u) | u \in E\}$ and Π_Δ is the orthogonal projection on the diagonal $\Delta = \{(a, a) | a \geq 0\}$.

The **Persistence Fisher Kernel** (**PFK**) is then defined as

$$K_{PF}(D,E) := \exp(-t d_{FIM}(D,E)),\ t>0, \text{ for any } D,E \in \mathfrak{D}.$$

Persistence Image (PI)

The main reference is [2]. If $D \in \mathfrak{D}$ we introduce a change of coordinates, $T : \mathbb{R}^2 \to \mathbb{R}^2$ given by $T(x,y) = (x, y-x)$ and let $T(D)$ be the transformed multiset in first-persistence coordinates. Let $\phi_u : \mathbb{R}^2 \to \mathbb{R}$ be a differentiable probability distribution with mean $u = (u_x, u_y) \in \mathbb{R}^2$, usually $\phi_u = g_u$, where g_u is the 2-dimensional Gaussian with mean u and variance σ^2, defined as

$$g_u(x,y) = \frac{1}{2\pi\sigma^2} e^{-[(x-u_x)^2+(y-u_y)^2]/2\sigma^2}.$$

Fix a weight function $f : \mathbb{R}^2 \to \mathbb{R}$, that is $f \geq 0$, it is equal zero on the horizontal axis, continuous and piecewise differentiable. A possible choice is a function that depends only on the persistence coordinate y, a function $f(x,y) = w_b(y)$ where

$$w_b(t) = \begin{cases} 0 & \text{if } t \leq 0, \\ \frac{t}{b} & \text{if } 0 < t < b, \\ 1 & \text{if } t \geq b. \end{cases}$$

Definition 30. Given $D \in \mathfrak{D}$, the corresponding **persistence surface** $\rho_D : \mathbb{R}^2 \to \mathbb{R}$ is the function

$$\rho_D(x,y) = \sum_{u \in T(D)} f(u)\phi_u(x,y).$$

If we divide the plane in a grid with n^2 pixels $(P_{i,j})_{i,j=1,\dots,n}$, we have the following

Definition 31. Given $D \in \mathfrak{D}$, its **persistence image** is the collection of pixels

$$PI(\rho_D)_{i,j} = \int\int_{P_{i,j}} \rho_D(x,y)dxdy.$$

Thus, through persistence image, each persistence diagram is turned into a vector $PIV \in \mathbb{R}^{n^2}$ that is $PIV(D)_{i+n(j-1)} = PI(D)_{i,j}$, then it is possible to introduce the following kernel

$$K_{PI}(D,E) =< PIV(D), PIV(E) >_{\mathbb{R}^{n^2}}.$$

4.3.2 Shape parameters analysis

Each aforementioned kernel has some parameters, that have been chosen through the cross-validation phase. As in the context of RBF as explained in [26], also in the Machine Learning framework, it is better to tune the parameters accordingly to the so-called trade-off principle, for instance, such that the condition number of Gram matrix is not so high and on the other hand the accuracy is satisfactorily high. The aim here is to run such analysis to kernels presented in this thesis.

The **PSSK** has only one parameter to tune σ. Typically the users consider $\sigma \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$. We run the CV phase for different shuffles of a dataset and plot the results in terms of the condition number of the Gram matrix related to the training samples and the accuracy. For our analysis, we consider $\sigma \in \{0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 500, 800, 1000\}$ and run tests on some datasets cited in the following. The results are similar in each case so we decided to report ones about SHREC14 dataset.

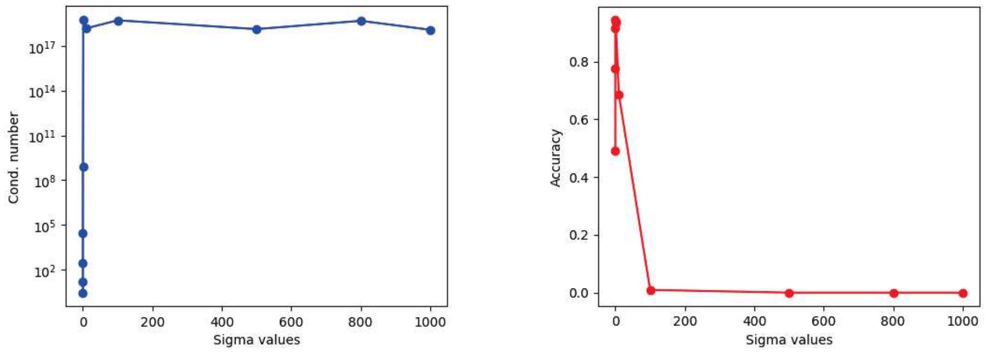


Figure 4.3: Comparison results about PSSK for SHREC14 in terms of condition number (left) and accuracy (right) with different σ

From the plot, it is evident how large values of σ bring to unstable matrix and less accuracy. Then in what follows, we will take into account only $\sigma \in \{0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10\}$.

PWGK is the kernel with a higher number of parameters to tune, then it is not so evident what are the best-set values to take into account. We choose reasonable starting sets as: $\tau \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$, $\rho \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$, $p \in \{1, 5, 10, 50, 100\}$, $C_w \in \{0.001, 0.01, 0.1, 1\}$. Due to a large number of parameters, we first ran some experiments varying (ρ, τ) with fixed (p, C_w), and then we reversed the roles.

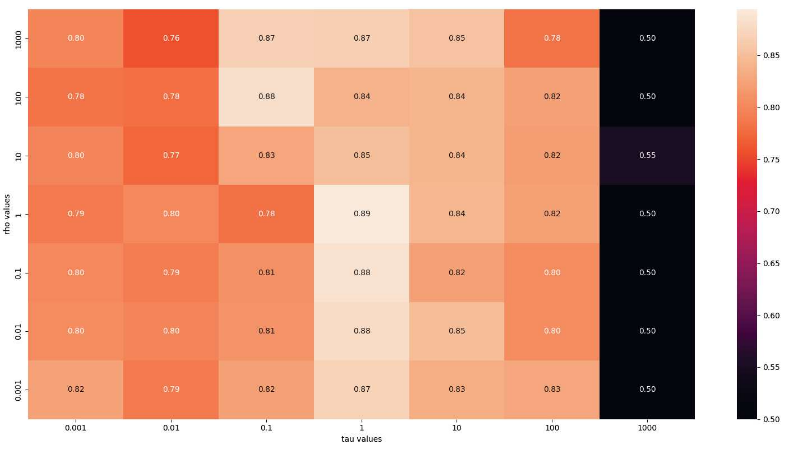


Figure 4.4: Comparison results about PWGK for MUTAG in terms of accuracy with different τ and ρ

We report here in Figure 4.4 only a plot for fixed C_w and p because it highlights how high values of τ (for example $\tau = 1000$) are to be excluded. We find this behavior for different values of C_w, p and various datasets, here the case $C_w = 1$, $p = 10$ and MUTAG dataset. Therefore we decide to vary the parameters as follows: $\tau \in \{0.001, 0.01, 0.1, 1, 10, 100\}$, $\rho \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$, $p \in \{1, 5, 10, 50, 100\}$, $C_w \in \{0.001, 0.01, 0.1, 1\}$. Unluckily there is no other evidence that can guide the choices, except for τ, where values $\tau = 1000$ always have bad accuracy, as one can see below in the case of MUTAG with shortest path distance.

In the case of **SWK**, there is only one parameter η. In [16], the authors propose to consider values starting from the first and last decile and the median value of the gram matrix of the training samples flatten in order to obtain a vector, then they multiply these three values for $0.01, 0.1, 1, 10, 100$. For our analysis, we have decided to study the behavior of such kernel considering the same set of values, independently from the specific dataset. We consider $\eta \in \{0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 500, 800, 1000\}$.

We run tests on some datasets and the plot, related to the DHFR dataset, reveals evidently that large values for η are to be excluded. So, we will decide to take η only in $\{0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10\}$.

PFK has two parameters: the variance σ and t. In [41], the authors exhibit

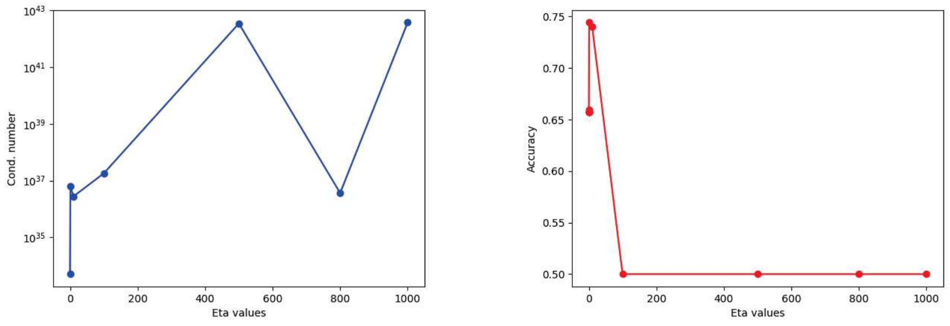


Figure 4.5: Comparison results about SWK for DHFR in terms of condition number (left) and accuracy (right) with different η

the procedure to follow in order to obtain the corresponding set of values. It shows that the choice of t depends on σ. Instead, our aim in this paper is to carry out an analysis that is dataset-independent and that turns out to be strictly connected only to the definition of kernel itself. First, we take different values for (σ, t) and we plot the corresponding accuracies, here in the case of MUTAG with shortest path distance, but the same behavior holds true also for other datasets.

The condition numbers are indeed high for every choice of parameters and therefore we avoid reporting here because it would be meaningless. From the plot, it is evident that it is convenient to set σ lower or equal to 10 instead t should be set bigger or equal to 0.1. Thus in what follows, we take into account $\sigma \in \{0.001, 0.01, 0.1, 1, 10\}$ and $t \in \{0.1, 1, 10, 100, 1000\}$.

In the case of **PI**, we considered a reasonable set of values for the parameter $\sigma \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$. The results are related to BZR with the shortest path distance.

As in the previous kernels, it seems that the accuracy is better for small values of σ. For this reason, we set $\sigma \in \{0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10\}$.

But then an interesting and obvious question arose: what is the "best" kernel to use? Is this goodness only related to the intrinsic mathematical structure of kernels or also to the kind of data, we want to classify?

Of course, we looked for the answer in the literature and we did not find a rigorous, complete, and unique treatment of the topic also regarding different kinds of data. From this lack, the idea to unify the main available results in a unique paper and to make a rigorous analysis, comparing performances as kernel and kind of data varies. For example, we have considered point cloud data, grey images, graphs, and signals

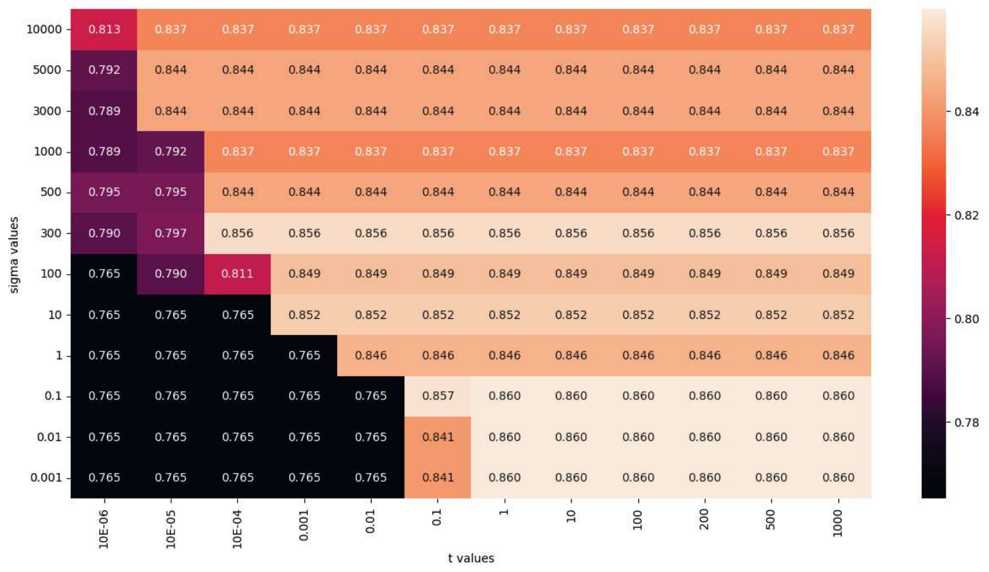


Figure 4.6: Comparison results about PFK for MUTAG in terms of accuracy with different t and σ

and the results are indeed similar.

4.3.3 Numerical Tests

For what concerns the computation of simplicial complexes and persistence diagrams, we use some Python libraries available online as `gudhi` [70], `ripser` [64], `giotto-tda` [71] and `persim` [57]. To all datasets, we have performed a random splitting (70%/30%) for training and testing and applied a 10-fold cross-validation on the training set for the hyperparameters tuning. Then we averaged the results over 10 runs.

In tests, we use the implementation of SVM provided by the `Scikit` [49] library of Python. For PFK, we precomputed the Gram matrices using a Matlab (Matlab R2023b) routine because it is faster than the Python one. The values for C belong to $\{0.001, 0.01, 0.1, 1, 10, 100\}$. For each kernel, we have considered the following values for the parameters:

- **PSSK**: $\sigma \in \{0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10\}$

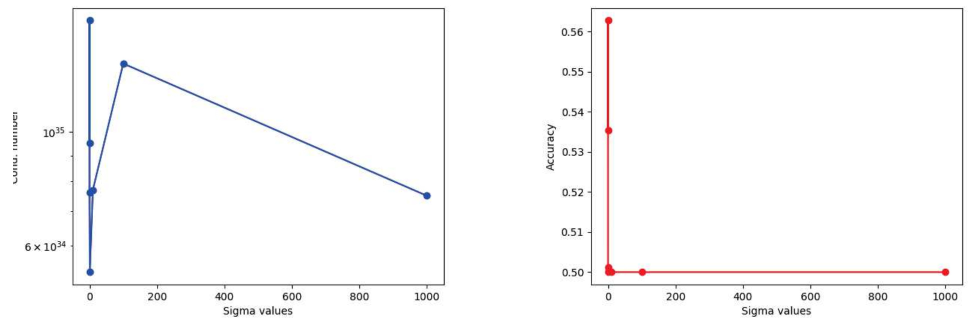


Figure 4.7: Comparison results about PI for BZR in terms of condition number (left) and accuracy (right) with different σ

- **PWGK**: $\tau \in \{0.001, 0.01, 0.1, 1, 10, 100\}$, $\rho \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$, $p \in \{1, 5, 10, 50, 100\}$, $C_w \in \{0.001, 0.01, 0.1, 1\}$ and for kernel we chose the Gaussian one.
- **SWK**: $\eta \in \{0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10\}$
- **PFK**: $\sigma \in \{0.001, 0.01, 0.1, 1, 10\}$ and $t \in \{0.1, 1, 10, 100, 1000\}$
- **PI**: $\sigma \in \{0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10\}$ and number of pixel 0.1.

All codes have been run using Python 3.11 on a 2.5 GHz Dual-Core Intel Core i5, 32 Giga RAM. They are available on the GitHub page

`https://github.com/cinziabandiziol/persistence_kernels`

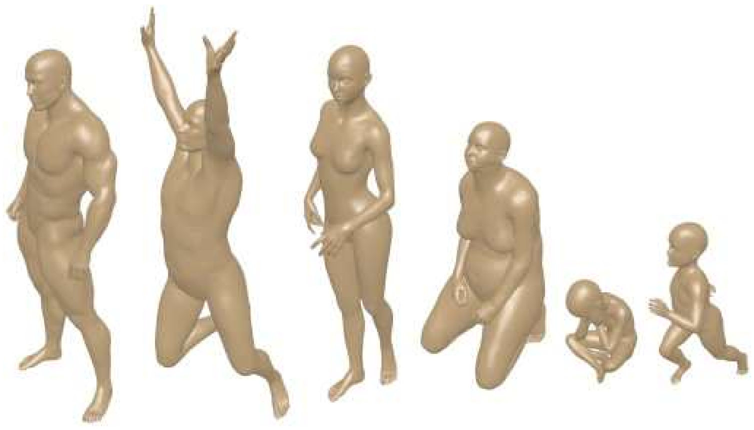
4.3.4 Point cloud data and shapes

Protein

This is the Protein Classification Benchmark dataset PCB00019 [62]. It sums up information for 1357 proteins corresponding to 55 classification problems. The data are highly imbalanced and therefore we apply the classifier to one of them, where the imbalance is slightly less evident. Persistence diagrams were computed for each protein by considering the 3-D structure or better the (x, y, z) position of any atoms in each of the 1357 molecules, as a point cloud in $\mathbb{R}^3$. Finally using `ripser` we compute the persistence diagrams only of dimension 1.

SHREC14 - Synthetic data

The dataset is related to the problem of non-rigid 3D shape retrieval. It collects exclusively human models in different body shapes and 20 poses. It consists of 15 different human models, about man, woman, and child, each with its own body shape. Each of these models exists in 20 different poses making the dataset composed of 300 models.



For each shape, the meshes are given with about 60000 vertices and, using the Heat Kernel Signature (HKS) introduced in [63], over different values of t_i as [55], we have computed the persistence diagrams of the induced filtration in dimensions 1.

Orbit recognition

We consider the dataset proposed in [2]. We take into account the linked twisted map, which models fluid flows. The orbits can then be computed through the following discrete dynamical system

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

where the starting point $(x_0, y_0) \in [0,1] \times [0,1]$ and $r > 0$ is a real parameter that influences the behavior of the orbits, as appears in images.

As in [2], $r \in 2.5, 3.5, 4, 4.1, 4.3$ and is strictly connected to the label of the corresponding orbit. For each of them, we compute the first 1000 points of 50 orbits,

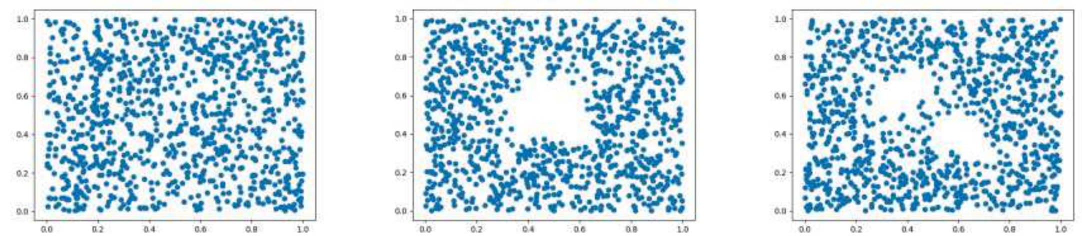


Figure 4.8: Orbits composed by the first 1000 iterations of the twisted map with $r = 3.5, 4.1, 4.3$ from left to right

with starting points chosen randomly. The final dataset is made up of 250 elements. We compute PD considering only the 1-dimensional features. Since each PD has a large number of topological features, we decided to consider only the first 10 most persistent, as done in [23].

Kernel	**PROTEIN**	**SHREC14**	**DYN SYS**
PSSK	**0.561**	0.933	0.829
PWGK	0.538	0.923	0.819
SWK	0.531	**0.935**	**0.841**
PFK	0.556	**0.935**	0.784
PI	0.560	0.934	0.777

Table 4.5: Accuracy related to point cloud and shape datasets

First, the high difference in performances through different datasets is probably due to the high imbalance of the PROTEIN one with respect to the perfect balance of the other ones. It is well known that, if the classifier does not have enough samples for each class, as in the case of the imbalanced dataset, it has to face high issues in classifying correctly elements of the minor classes. Except for PROTEIN where the PSSK shows slightly better performances, for SHREC14 and DYN SYS the best accuracy has been achieved by SWK.

4.3.5 Images

All the definitions introduced in Section 2 can be extended to another kind of simplicial complex, the cubical complex. It is useful when, for example, one deals with images or objects based on meshes, for example. More precise from [43]

Definition 32. An elementary cube $Q \subset \mathbb{R}^d$ is defined as a product $Q = I_1 \times \cdots \times I_d$ where each I_j is either a singleton set $\{m\}$ or a unit length interval $[m; m+1]$ for some integers $m \in \mathbb{Z}$. The number k of unit length intervals in the product of Q is called the dimension of cube Q, and we call Q a k-cube. If Q and $\bar{Q}$ are two cubes and $Q \subset \bar{Q}$, then Q is said to be a face of $\bar{Q}$. A cubical complex X in $\mathbb{R}^d$ is a collection of k-cubes $(0 \leq k \leq d)$ such that:

- every face of a cube in X is also in X;
- the intersection of any two cubes of X is either empty or a face of each of them.

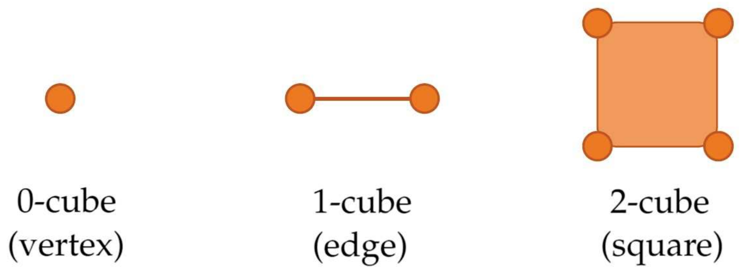


Figure 4.9: Cubical simplices

MNIST and FMNIST

MNIST [42] is very common in the classification framework. It consists of 70000 handwritten digits, in grayscale, which one could try to classify into 10 different classes. Each image can be viewed as a set of pixels with a value between 0 and 256 (black and white) as in the figure.

Starting from this kind of dataset, we have to compute the corresponding persistent features. According to the approach proposed in [30] coming from [7], we first binarize each image, for instance, we replace each grayscale image with a white/black one, then we use as filtration function the so-called **Height filtration** $\mathcal{H}(p)$ in [30]. For cubical complex, for a chosen vector $v \in \mathbb{R}^d$ of unit norm, it is defined as

$$\mathcal{H}(p) = \begin{cases} \langle p, v \rangle & \text{if p is black,} \\ H_\infty & \text{otherwise} \end{cases}$$

where H_∞ is a large default value chosen by the user. As in [7] we have chosen 4 different vectors for p: $(1,0), (-1,0), (0,1), (0,-1)$ and we have computed 0 and

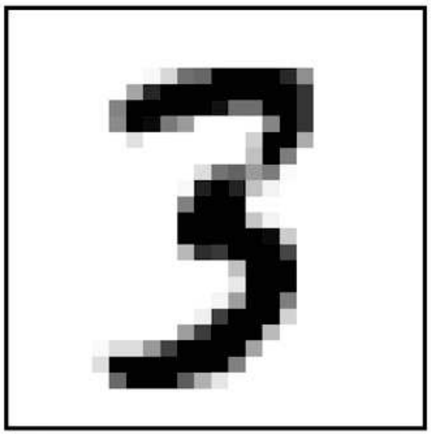


Figure 4.10: Example of an element in MNIST dataset

1-dimensional persistent features using the `tda-giotto` and `gudhi` libraries. Finally, we concatenate them. For the current experiment, we decided to focus the test on a subset of the original MNIST, composed of only 10000 samples. This is a balanced dataset. Due to some memory issues, for this dataset we have to consider a pixel size of 0.5 and for PWGK only $\tau \in \{0.001, 0.01, 0.1, 1, 10, 100\}$, $\rho \in \{0.001, 0.1, 10, 1000\}$, $p = 10$, $C_w \in \{0.001, 0.01, 0.1, 1\}$.

Another example of a grayscale image dataset is the FMNIST [66], which contains 28 x 28 gray-scale images related to the fashion world.

To deal with it, we follow another approach proposed in [4], where the authors apply padding, median filter, shallow thresholding, and canny edges, and then compute the usual filtration to the obtained image. Due to some memory issues, for this dataset we have to consider a pixel size of 1 and for PWGK only $\tau \in \{0.001, 0.01, 0.1, 1, 10, 100\}$, $\rho \in \{0.001, 0.1, 10, 1000\}$, $p = 10$, $C_w \in \{0.001, 0.01, 0.1, 1\}$.

Kernel	**MNIST**	**FMNIST**
PSSK	0.729	0.664
SWK	**0.802**	**0.709**
PWGK	0.754	0.684
PFK	0.734	0.671
PI	0.760	0.651

Table 4.6: Accuracy related to MNIST and FMNIST

Figure 4.11: Example of an element in FMNIST dataset

Both datasets are balanced and probably the results are better in the case of MNIST due to the fact that it is easier to classify handwritten digits instead of images of cloths. SWK shows slightly better performances.

4.3.6 Graphs

In many different contexts, from medicine to chemistry, data can have the structure of graphs. Graphs are couples of the set (V, E) where V is the set of vertices and E is the set of edges. The graph classification is the task of attaching a label/class to each whole graph. In order to compute the persistent features, we need to build a filtration. In the context of graphs, as in other cases, there are many different definitions, see for example [3].

We consider the Vietoris Rips filtration, where starting from the set of vertices, at each step we add the corresponding edge whose weights are less or equal to a current value ϵ. This turns out to be the most common choice and the software available online allows us to build it after providing the corresponding adjacency matrix. In our experiments, we consider only undirected graphs but, as in [3], building a filtration is possible also for directed graphs. Once defining the kind of filtration to use, one needs again to choose the corresponding weights. We decide to take into account: first, the shortest path distance and then the Jaccard index as for example in [68].

Given two vertices $u, v \in V$ the **shortest path distance** is defined as the minimum number of different edges that one has to meet going from u to v, or vice versa since the graphs here are considered as undirected. In graphs theory, it is a widely use metric.

Instead, the Jaccard index is a good measure of edge similarity. Given an edge $e = (u, v) \in E$ then the corresponding **Jaccard index** is computed as

$$\rho(u, v) = \left| \frac{NB(u) \cap NB(v)}{NB(u) \cup NB(v)} \right|$$

where $NB(u)$ is the set of neighbors of u in the graph. This metric recovers local information of nodes in the sense that two nodes are considered similar if their neighbor sets are similar.

In both cases, we consider the sub-level set filtration and we collect 0 and 1 dimensional persistent features both.

We take 6 of such sets among the graph benchmark datasets, all undirected. They are

- **MUTAG**: it is a collection of nitroaromatic compounds and the goal is to predict their mutagenicity on Salmonella typhimurium
- **PTC**: is a collection of chemical compounds represented as graphs that report the carcinogenicity of rats
- **BZR**: it is a collection of chemical compounds and one has to classify them as active or inactive
- **ENZYMES**: it is a dataset of protein tertiary structures obtained from the BRENDA enzyme database and the aim is to classify each graph into 6 enzymes.
- **DHFR**: it is a collection of chemical compounds and one has to classify them as active or inactive
- **PROTEINS**: in each graph nodes represent the secondary structure elements and the task is to predict whether a protein is an enzyme or not.

Their properties are summarized in table 4.7, where the IR index is the so-called **Imbalanced Ratio** (**IR**), which denotes the imbalance of the dataset. (For more details see Appendix.)

Dataset	**N° Graphs**	**N° classes**	**IR**
MUTAG	188	2	125:63
PTC	344	2	192:152
BZR	405	2	319:86
ENZYMES	600	6	100:100
DHFR	756	2	461:295
PROTEINS	1113	2	663:450

Table 4.7: Graph datasets

Computations of the adjacency matrix and PDs are made using functions implemented in `tda-giotto`.

The performances achieved with two edge weights are reported in tables,

Kernel	**MUTAG**	**PTC**	**BZR**	**DHFR**	**PROTEINS**	**ENZYMES**
PSSK	0.868	**0.545**	0.606	0.557	0.668	0.281
PWGK	0.858	0.510	0.644	0.655	**0.694**	0.329
SWK	**0.872**	0.511	**0.712**	**0.656**	0.686	**0.370**
PFK	0.842	0.534	0.682	**0.656**	**0.694**	0.341
PI	0.863	0.542	0.585	0.519	0.691	0.285

Table 4.8: Balanced Accuracy related to graph datasets using shortest path distance (Accuracy only for ENZYMES dataset)

It is evident how SVM with any PKs can't be good at classifing ENZYMES. At the moment no motivation has been found to justify this behaviour. Thanks to these results two conclusions can be taken. The first one is that, as expected, the performance of the classifier depends on the particular filtration used for the computation of persistent features. The second one is related to the fact that SWK and PFK seem to work slightly better than the other kernels: in the case of shortest path distance, SWK is to be preferred while PFK seems to work better in the case of the Jaccard index. In the case of PROTEINS, in both cases, PWGK provides best-balanced accuracy.

Kernel	**MUTAG**	**PTC**	**BZR**	**DHFR**	**PROTEINS**	**ENZYMES**
PSSK	0.865	0.490	0.704	0.717	0.675	0.298
PWGK	0.859	0.516	**0.720**	0.727	**0.699**	0.355
SWK	0.858	0.523	0.703	0.726	0.689	**0.406**
PFK	**0.874**	**0.554**	0.704	**0.743**	0.678	0.400
PI	0.846	0.478	0.670	0.712	0.690	0.280

Table 4.9: Balanced Accuracy related to graph datasets using Jaccard Index (Accuracy only for ENZYMES dataset)

4.3.7 1-Dimensional Time Series

In many different applications, one can deal with 1-dimensional time series, see for examples [36], [50], [40], [19]. A 1-dimensional time series is a set $\{x_t \in \mathbb{R} | t = 1, \dots, T\}$. In [54] authors provide different approaches to build a filtration upon this kind of data. We decide to adopt the most common one. Thanks to the Taken's embedding, these data can be translated into point clouds. With suitable choices for two parameters: $\tau > 0$ the delay parameter and $d > 0$ the dimension, it is possible to compute a subset of points in $\mathbb{R}^d$ composed by $v_i = \{x_i, x_{i+\tau}, \dots, x_{i+(d-1)\tau}\}$ for $i = 1, \dots, T-(d-1)\tau$. The theory mentioned above related to point clouds can now be applied to signals, as points in $\mathbb{R}^d$. For how to choose values for the parameters, see [54]. The dataset for tests is taken from the UCR Time Series Classification Archive (2018) [21], which consists of 128 datasets of time series from different worlds of application. In the archive there is the splitting into test and train sets but, for the aim of our analysis, we don't take care of it and we consider train and test data as a whole dataset and then codes provide properly the subdivision.

Dataset	**N° time series**	**N° classes**	**IR**
ECG200	200	2	133:67
SONY	621	2	349:272
DISTAL	876	2	539:337
STRAWBERRY	983	2	632:351
POWER	1096	2	549:547
MOTE	1272	2	685:587

Table 4.10: Time series datasets

Using `giotto-tda` we compute the persistent features of dimensions 0,1,2 and join them together. The final results of the datasets are reported here.

Kernel	**ECG200**	**SONY**	**DISTAL**	**STRAW.**	**POWER**	**MOTE**
PSSK	0.642	0.874	0.658	0.814	0.720	0.618
PWGK	0.726	0.888	0.696	0.892	0.769	0.633
SWK	**0.731**	0.892	**0.723**	**0.898**	**0.784**	**0.671**
PFK	0.707	**0.895**	0.676	0.892	0.750	0.652
PI	0.717	0.841	0.662	0.793	0.712	0.606

Table 4.11: Balanced Accuracy related to time series datasets

As in the previous examples, SWK is winning and provides slightly best performances in terms of accuracy.

We have compared the performance of five Persistent Kernels applied to data of different natures. The results show how different PK are indeed comparable in terms of accuracy and there is not a PK that emerges clearly above the others. However, in many cases, the SWK and PFK perform slightly better. In addition, from a purely computational point of view, SWK is to be preferred since, by construction, the preGram matrix is parameter-independent. Therefore, in practice, the user has to compute such a matrix on the whole dataset only once at the beginning and then choose a suitable subset of rows and columns to perform the training, cross-validation, and test phases. This aspect is relevant and reduces the computational costs and time compared with other kernels.

Kernel	**SHREC14**	**BZR**	**DISTAL**
PSSK	582	11731	49481
PWGK	1841	3751	43152
SWK	**209**	**321**	**1418**
PFK	319	814	11405
PI	266	640	1825

Table 4.12: Computational costs (seconds) marked in **bold** the best values

For sake of completeness, we collect in Table 4.12 the time needed by the algorithm to solve the classification problem using different kernels. We consider, as example, 3 datasets among those used in the previous tests. As expected SWK turns out to be less expensive than the other kernels in terms of time consuming.

Another aspect to be considered, as in the case of graphs, is how to choose the function f that provides the filtration. The choice of such a function is still an open problem and an interesting field of research. The right choice in fact would guarantee to be able to better extract the intrinsic information from data, improving, in this way, the classifier's performances. For the sake of completeness, we recall here that in the literature there is also an interesting direction of research whose aim is to build a new PK starting from the main 5. From one of the PKs mentioned in previous sections, the authors in [23] studied how to modify them obtaining the so-called **Variably Scaled Persistent Kernels**, which are Variably Scaled Kernels applied to the classification context. The results reported by the authors are indeed promising, thus it could be another interesting direction for further analysis.

As final remarks we observed, as expected and evident in Table 4.11, the goodness of the model in the classification framework using SVM is independent of the data structure and the best kernel seems to be the SWK that has been revealed winning also from the computational point of view. These results have been published in [6].

Chapter 5

Intrinsic Dimension with PH

The concept of intrinsic dimension is important in various fields, particularly in Machine Learning, data analysis and geometry, or in general where the complexity and structure of the data need to be understood and leveraged effectively. In high-dimensional data analysis, manifold learning, data compression, or Machine Learning, understanding the true dimensionality of the data is crucial to optimize processing and reveal meaningful patterns. We start our analysis with two examples that intuitively give the idea behind the concept of **Intrinsic Dimension (ID)**.

In physics, scientists deal with models that usually depend on D degrees of freedom or parameters. But, usually, some of them are useless or redundant and can be removed. Thus ID represents, roughly speaking, the minimal and true number of parameters needed for the model to faithfully represent the phenomenon under analysis. Identifying the Intrinsic Dimension helps physicists to simplify models and understand the fundamental behaviors of complex systems, such as in statistical mechanics or quantum mechanics. On the other hand, also in the context of Machine Learning, estimating ID is very fruitful. Managing a large amount of data with a large number D of features is today a real problem and a challenge to face. Having the ability to estimate a value that counts the number of essential and fundamental features, $d << D$, would also be crucial to understand the structure of the data. Finally, identifying the ID allows for more efficient data representation, reducing noise, improving model performance, and making data more treatable also from a computational point of view. Working in a lower-dimensional space often leads to faster algorithms and less resource-intensive computations.

More formally from [12],

Definition 33. A dataset $\Omega \subset \mathbb{R}^D$ is said to have **Intrinsic Dimension (ID)** equal to d if its elements lie entirely, without loss of information, within a d - dimensional manifold $\mathcal{M}$ of $\mathbb{R}^D$, where $d < D$.

In both previous contexts, it is clear the reason why many efforts have been made and continue to be made in research also today to define a suitable algorithm able to extract a faithful approximation, or better, estimation of the ID.

5.0.1 ID estimators and Dimensionality Reduction

It is interesting to point out how finding a good ID estimator is a real challenge since the first algorithm for estimating data dimensionality dates back to 1969 and this is still an open problem of research [12]. Today in the literature, there are many ID estimators. As well explained in [12], such estimators can be divided into three groups:

- **GLOBAL**: they are based on the assumption that points lie on a unique manifold with fixed dimension
- **LOCAL**: they provide an estimation on ID for each small subset of data
- **POINTWISE**: provide an estimation of ID of each pattern and averaging they give an approximation of global ID

We cite only some famous and interesting examples of Global methods, which are the Principal Component Analysis (PCA) with its variants and Fractal-based methods.

Fractal-based methods for estimating Intrinsic Dimension focuses on understanding how complex structures (often called *fractals*) are embedded in higher-dimensional spaces. These methods use the concept of fractals, which are irregular and self-similar structures, to estimate the true underlying dimension of a dataset. The key idea behind fractal-based methods is that many real-world datasets lie on fractals or lower-dimensional manifolds in a high-dimensional space, and estimating the fractal dimension can help understand the intrinsic dimension of the data. The most famous examples and milestones are the Box counting dimension and Correlation dimension.

The **Box-Counting** is another well-known fractal-based technique that estimates the fractal dimension. It involves covering the data points with boxes of a certain size and counting how many boxes are required to cover the dataset. By reducing the box size, you observe how the number of boxes changes. The scaling behavior of the number of boxes needed to cover the dataset is then analyzed. The exponent that describes this scaling gives the ID,

$$N(\epsilon) \propto \epsilon^d$$

where $N(\epsilon)$ is the number of boxes needed to cover the data at scale ϵ and d is the fractal dimension.

Exactly, from [1],

Definition 34. Let Ω be a subset of $\mathbb{R}^D$, considered as a metric space, and let N_ϵ denote the infimum of the number of closed balls of radius ϵ required to cover Ω. Then the **Box-Counting Dimension** of Ω is

$$d = \lim_{\epsilon \to 0} \frac{\log(N_\epsilon)}{\log(1/\epsilon)}$$

provided this limit exists. Relaxing the limit with a lim sup gives the upper box-counting dimension, and a lim inf gives the lower box-counting.

In applications, another ID estimator has become the most common and used. This is the case of the *Correlation Dimension*.

The Correlation Dimension is a concept used in the field of fractal geometry and chaos theory to quantify the complexity of a set of points in a space. It provides a way to measure how the number of points in a dataset scales with the size of the space in which they are distributed. Essentially, it helps to determine how "dense" or "spread out" the points are as you look at them at different scales. A higher correlation dimension indicates a more complex structure, while a lower dimension suggests a simpler arrangement. It is often used in analyzing time series data and understanding the underlying patterns in chaotic systems.

The Correlation Dimension captures how the number of points within a certain distance (or radius) scales with the radius as it increases. If the data points lie on a fractal, the number of points within a given radius will grow in a power-law manner. Mathematically, it is defined using the correlation integral, which measures the probability that pairs of points in a dataset are within a certain distance r of each other.

From [1],

Definition 35. Let Ω be a subset of $\mathbb{R}^D$ equipped with a measure μ and let $\mathcal{X}_n$ be a random sample of n points in Ω. Let $H : \mathbb{R} \to \mathbb{R}$ the Heaviside step function, or more precisely $H(x) = 0$ for $x < 0$ and $H(x) = 1$ otherwise. Fixed $r > 0$, the correlation integral of μ is defined to be

$$C(r) = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} H(r - \|x_i - x_j\|).$$

The **Correlation Dimension** D is defined as the slope of the log-log plot of $C(r)$ versus r in the limit as r approaches zero:

$$d = \lim_{r \to 0} \frac{\log C(r)}{\log(1/r)}$$

The idea behind the definition is that, as $r \to 0$ the correlation integral scales as

$$C(r) \propto r^d$$

where d is the correlation dimension (intrinsic dimension estimate).

In application, given a finite set of points $\{x_1, \ldots, x_n\}$ the **Correlation dimension** can be computed as

$$\lim_{\epsilon \to 0} \lim_{n \to +\infty} \frac{\log(C(n, \epsilon))}{\log(\epsilon)}$$

where

$$C(n, \epsilon) = \frac{\#\{(x_i, x_j) : d(x_i, x_j) < \epsilon, i < j\}}{n(n-1)}.$$

Even if the estimation of Intrinsic Dimension is essential for example in quantifying the complexity of the data in terms of the minimal number of dimensions required to capture the data's variance, at the same time in application having at hand methods able to transform the original high-dimensional data into a lower-dimensional representation is crucial. These techniques go under the name of **Dimensionality Reduction Methods**. The main goal is to transform the data into a lower-dimensional space while preserving the relevant information or structure of the data, such as variance, similarity, or class separability. This competes to simplify the data, enhance computational efficiency, reduce noise, and find patterns that might not be visible in higher dimensions. Especially, in Machine Learning and Data Analysis, it can help improve model performance, reduce computation time, and mitigate the curse of dimensionality. A lot of different techniques have been proposed in the literature in the last twenty years and more. Without any doubts, the most famous are **t-SNE** and **UMAP**.

t-SNE, or better t-Distributed Stochastic Neighbor Embedding, is particularly effective for visualizing high-dimensional data in two or three dimensions. It focuses on preserving the local structure of the data, making it useful for exploring clusters and patterns. But it has revealed some drawbacks and so a new method has emerged as winning and today most used, the **UMAP**.

UMAP, or Uniform Manifold Approximation and Projection, is a powerful technique for dimensionality reduction that is particularly effective for visualizing high-dimensional data. It is similar to t-SNE but often faster and capable of preserving more of the global structure of the data. It is widely used in fields like bioinformatics, image analysis, and natural language processing for tasks such as clustering, visualization, and feature extraction. In addition, it aims to preserve both local and global structures in the data, it is generally faster than t-SNE, especially on larger datasets, and can be used for various types of data, including continuous, categorical, and mixed data types. It can also be applied to different tasks, such as clustering and classification, and has only a few parameters that can be adjusted. These are only some advantages that make the method commonly used today in applications. Overall, UMAP is a versatile and efficient tool for dimensionality reduction, making it a popular choice among data scientists and researchers. From our point of view, an interesting aspect that we want here to underline is its strong connection with the field of algebraic topology. In fact, it leverages concepts from this field to understand the structure of data.

UMAP is based on the idea that high-dimensional data often lies on or near a lower-dimensional manifold. To make the so-called *Manifold Learning*, concepts introduced in Chapter 3 have been taken into account. To study the properties of these manifolds, such as their shape and connectivity, it uses topological concepts as simplicial complexes, data points are treated as vertices of a graph, and connections (edges) are formed based on the local neighborhood of each point. It aims to preserve topological features of the data, such as clusters and holes when projecting it into a lower-dimensional space. By maintaining these features, UMAP can reveal important patterns and structures that might be lost in other dimensionality reduction techniques. The idea is to capture the multi-scale structure of the data, which is a key aspect of understanding its topology, more or less as in the case of Persistent Homology. In summary, UMAP utilizes concepts from algebraic topology to create a meaningful representation of high-dimensional data, focusing on the underlying manifold structure and topological features. This connection allows UMAP to be a powerful tool for visualizing and analyzing complex datasets. Regarding the theory of TDA and mainly of Persistent Homology and inspired by the promising result of UMAP that combines topological concepts, we are interested in investigating if it could be possible to create a good estimator for ID using PH. After a deep analysis of the bibliography available, we end up with the available definitions: **i-Dimensional**

Persistent Homology Fractal Dimension [1], **Persistent Homology Dimension** [35] and **Persitent Homology Complexity** [35].

5.1 ID estimators using PH

First going into details we recall here a concept useful for the following definition. We assume to have a metric space X or a Manifold $\mathcal{M}$ immersed in some $\mathbb{R}^D$ equipped with a probability measure μ. Let $\mathcal{X}_n$ denote a set of n points sampled from X ($\mathcal{M}$) according to μ, then, for any $\alpha > 0$, we introduce

$$E^i_\alpha(\mathcal{X}_n) := \sum_{I \in PH_i(\mathcal{X}_n)} |I|^\alpha$$

where $PH_i(\mathcal{X}_n)$, $i = 0, 1, 2, \dots, D$ indicate the collections of topological features with dimensions $0, 1, 2, \dots, D$ and $|I|$ denotes the persistence, or lifetime, of topological feature I.

5.1.1 i-dim. PHFD

This is the first ID estimator to use persistent features proposed in the literature. In the related paper [1], the authors, starting from the definition of baseline methods, suggest their own definition for an ID estimator, that is

Definition 36. Let X be a metric space equipped with a probability measure μ, let $\mathcal{X}_n \subset X$ be a random sample of n points from X distributed according to μ, and let $E^i_1(\mathcal{X}_n)$ as above. The **i-dimensional Persistent Homology Fractal Dimension (i-dim. PHFD)** of μ is given by

$$dim^i_{PH}(\mu) = \inf_{d>0}\{d | \exists C(i, \mu, d) : E^i_1(\mathcal{X}_n) \leq C n^{(d-1)/d} \text{ with probability 1 as n} \to +\infty\}.$$

Although the authors of [1] were unable prove it, they formulated the following conjecture,

Conjecture: Let μ be a probability measure on a compact set $X \subset \mathbb{R}^d$ with $d \geq 2$ and let μ be nonsingular. Then for all $0 \leq i < d$, there is a constant $C \geq 0$ (depending on μ, m, and i) such that $E^i_1(\mathcal{X}_n) = C n^{(d-1)/d}$ with probability one as $n \to +\infty$.

If it was true, it would mean that, taking the logarithm of both parts,

$$\log(E_1^i(\mathcal{X}_n)) = \frac{d-1}{d}\log(n) + \log(C)$$

with d the ID that they want to estimate. This means that, in order to extract an approximation of d, one has to compute the linear approximation of points $(\log(n), \log(E_1^i(,\mathcal{X}_n)))$, and the corresponding slope of the line will be an approximation of $\frac{d-1}{d}$, ending then with a simple inversion to the final value d. In application, such results can be seen through the Log-Log plot.

5.1.2 PH dimension

Another interesting definition that of course takes inspiration from the previous one is the PH dimension. Considering the same notations, in [35] the authors define

Definition 37. Let X be a bounded subset of a metric space and μ a measure defined on X. For each $i \in \mathbb{N}$ and $\alpha > 0$, we define the **Persistent Homology dimension (PH dim)** as

$$dim_{PH_i^\alpha}(\mu) = \frac{\alpha}{1-\beta}$$

where

$$\beta = \limsup_{n \to +\infty} \frac{\log(\mathbb{E}(E_\alpha^i(x_1, \ldots, x_n)))}{\log(n)}$$

Unfortunately, how to compute the mean (expected) value is not obvious in practise. In any case, analyzing in depth this new estimator, it appears to be an "extension" of i-dim. PHFD also checking how the authors have computed effectively such an estimator. Observing their numerical results, it is also clear how the presence of the parameter α seems to increase its global performance.

5.1.3 i-dim. α PHFD

Inspired by these two definitions, we have decided to combine them, to hopefully obtain another good and interesting estimator, called **i-dim. α PHFD**, defining as

Definition 38. Let X be a metric space equipped with a probability measure μ, let $\mathcal{X}_n \subset X$ be a random sample of n points from X distributed according to μ, and let $E_1^i(X_n)$ as above. The **i-dimensional α Persistent Homology Fractal Dimension (i-dim. α PHFD)** of μ is given by

$$dim^{i,\alpha}_{PH}(\mu) = \inf_{d>0}\{d|\exists C(i,\mu,d) : E^i_\alpha(X_n) \leq Cn^{(d-1)/d} \text{ with probability 1 as n} \to +\infty\}.$$

5.1.4 PH complexity

On the other hand, another definition tries to measure the complexity of data, from this the name **PH complexity**.

Definition 39. If X is a subset of a metric space, we define the cumulative PH_i curve F_i by

$$F_i(X,\epsilon) = \#\{I \in PH_i(X)||I| > \epsilon\}$$

using the notation previously introduced. Then the PH_i complexity of X is given by

$$comp_{PH_i}(X) = \lim_{\epsilon\to 0}\frac{-\log(F_i(X,\epsilon)}{\log\epsilon}$$

As in the context of kernels, here we found again the lack of a comparison with different definitions and so we started testing them on some benchmark datasets and some datasets taken by the world of Neuroscience.

5.2 Numerical Tests

We have written down the estimators in our Python implementations. All codes have been run using Python 3.11 on a 2.5 GHz Dual-Core Intel Core i5, 32 Giga RAM. They are available in the GitHub page

```
https://github.com/cinziabandiziol/Topological_ID_Estimator
```

We have run our experiments by considering several datasets of different nature:

- **Benchmark Dataset**: these datasets are very common in the context of ID estimators. They consist of data sampled from ”regular” manifolds. We collect all the related information in Table below (see [13])

Dataset	**d**	**Description**
Helix	2	2-dimensional helix in $\mathbb{R}^3$
Swiss	2	Swiss-Roll in $\mathbb{R}^3$
Sphere	3	3-dimensional sphere linearly embedded in $\mathbb{R}^4$
NonLinear	4	Nonlinear Manifold in $\mathbb{R}^8$
Affine3d5d	3	Affine space in $\mathbb{R}^5$
Mist	4	Conc. figure, mistakable with a 3-dim. one in $\mathbb{R}^6$
CurvedManifold	12	Nonlinear (highly curved) manifold in $\mathbb{R}^{72}$
NonLinear6d36d	6	Nonlinear manifold in $\mathbb{R}^{36}$

- **Fractal Dataset**: a dataset from the world of fractals and dynamical systems, in particular Sierpinski and Ikeda Attractor (see Figure 5.2). For memory issues in computing topological features, here we have computed 4000 points for the Sierpinski Triangle and 5000 for the Ikeda Attractor respectively.

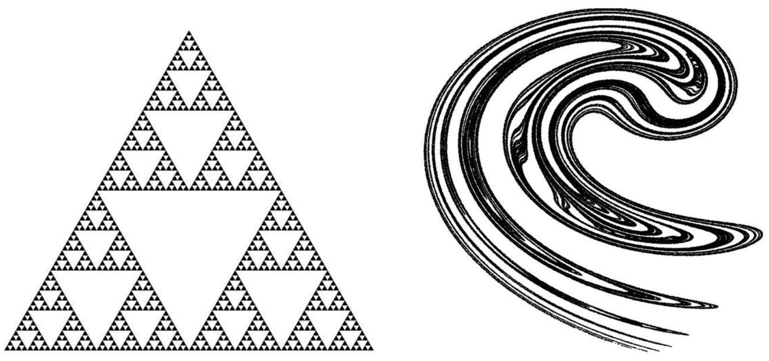


Figure 5.1: Sierpinski triangle (left) and Ikeda map (right)

- **Neuroscience Dataset**: it comes from a research project still open. Starting from the analysis of activity trajectories of particular recurrent neural networks (RNNs), the aim is to mirror the brain functionality related to basic tasks using a NN. In particular, we considered the RNNs developed by Yang et al. [67] for the study of the properties of the network while performing cognitive tasks. In the literature, the authors trained RNNs to solve simple tasks that mimic typical stimulus-response mapping in experiments with primates. They selected 20 interrelated tasks, useful to understand basic cognitive processes such as working memory, inhibition, and context-dependent integration. In each task, the network receives one or two inputs or 'stimuli', both representing an angular variable or direction (in real experiments, animals are shown dots moving in a specific direction in their left or right eye). In addition, the network receives a 'fixation input', corresponding to the instruction to maintain the gaze fixed on a cross in the center of the screen. When the fixation input disappears, the network should produce an 'output', representing a motor response. The correct output depends (in more or less simple ways) on the preceding stimuli (in real experiments, animals should move their gaze in a direction that is a function of the received stimuli). The figure represents the idea of how to use an RNN. We have considered only 3 stimuli, whose data are stored in .csv file with the name Fdgo, Context, and Reactgo_filtered with 25200, 1040,0 and 5200 respectively in $\mathbb{R}^{256}$.

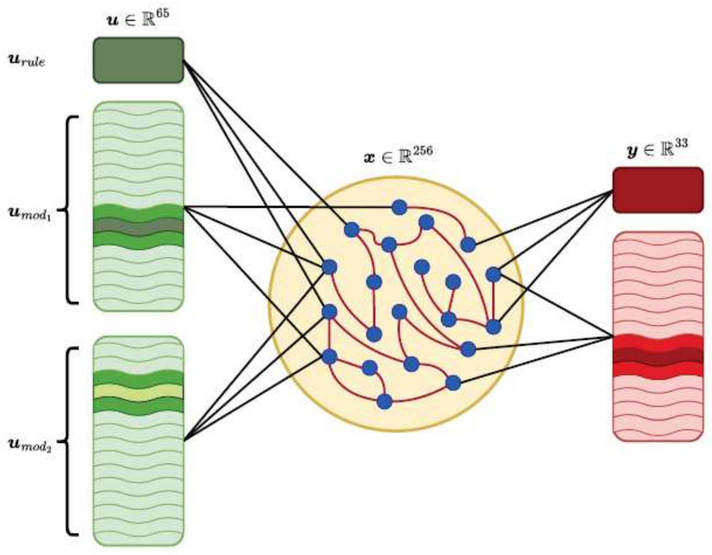

$\boldsymbol{u} \in \mathbb{R}^{65}$
$\boldsymbol{u}_{rule}$
$\boldsymbol{u}_{mod_1}$
$\boldsymbol{u}_{mod_2}$
$\boldsymbol{x} \in \mathbb{R}^{256}$
$\boldsymbol{y} \in \mathbb{R}^{33}$

Figure 5.2: Scheme of the source of data

For the last examples of datasets, coming from an open field of research, a priori the ID is completely unknown. To have only a probably decent idea of which ID is expected, we have decided to compute the related Correlation Dimension that, nowadays, is indeed such a good indicator in practice. Of course, for the sake of completeness, we finally have computed it for all datasets and collect here the results in Table 5.2, where d denotes the ID to approximate.

Dataset	d	Corr. dim.
Helix	2	1.99
Swiss	2	1.98
Sphere	3	2.98
NonLinear	4	3.87
Affine3d5d	3	3.01
Mist	4	3.54
CurvedManifold	12	11.66
NonLinear6d36d	6	5.82
Sierpinski Triangle	1.585	1.585
Ikeda	unk	1.68
Fdgo	unk	1.07
Contextdm1	unk	1.14
Reactgo new	unk	2.15

Table 5.1: Correlation Dimension of datasets

5.2.1 Shape Parameter Analysis

Taking the new definition, namely that of i-dim. α PHFD, it is evident how it depends on a parameter. Now we are interested in investigating which is the best choice for the parameter α. We have run an analysis similar to that made in the case of Persistence Kernels (see Chapter 3). Inspired by paper [35], we consider to take α, the parameter to test, in $(0, 4)$. We have chosen values not so large since the analysis in [35] has taken this direction that has revealed to be winning. For each value of α we computed the related i-dim. α PHFD. About the single computation of i-dim. α PHFD we have proceeded as follows:

1. Compute the persistent features, usually only of dimension $0, 1$
2. To mirror the limit $n \to \infty$, consider some numbers n_k of points closer to the maximum n available (e.g. for benchmark datasets $n = 10000$)
3. Take points as $(\log(n_k), \log(E^i_\alpha(\mathcal{X}_n)))$, compute the linear approximation using `numpy.polyfit` and then the slope is equal to $\frac{d-1}{d}$
4. Make the inverse and obtain the approximation d

We apply this workflow to all datasets with known ID and we have considered the approximation errors in computing the differences between the real ID and the approximated one. It exists an α such that this error is closer or equal to zero and this value will be the optimal one.

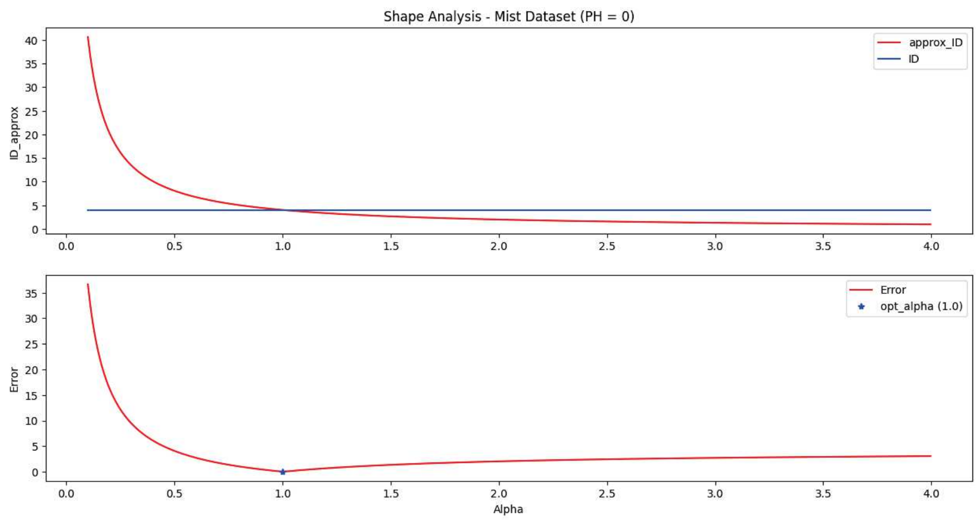


Figure 5.3: Shape Analysis Mist with PH_0

Interestingly, experiments reveal without any doubts that, considering both PH_0 and PH_1, the optimal value is exactly 1, revealing how the i-dim. PHFD is indeed optimal in α. We report in Figure 5.3 two plots: on the top the approximation ID (red) and the real ID (blue) and on the bottom we plot the errors and we have marked in blu the optimal point. The same behavior has been observed for all benchmark datasets.

5.3 ID estimates

First, we report here the approximation of ID of all datasets using the, now "optimal", definition of i-dim. PHFD.

It is immediately evident that, in general, considering PH_0, the estimator is able to perform better than using PH_1. On the other hand, this definition seems to meet some issues in the case of the Ikeda attractor when it was revealed to be completely far from the expected ID. On the other hand, if we consider the approximation of ID for Neuroscience datasets obtained using Correlation Dimension, again here the

Dataset	**d**	PH_0	PH_1
Helix	2	2.01	2.38
Swiss	2	1.93	2.16
Sphere	3	2.90	3.14
NonLinear	4	3.98	6.45
Affine3d5d	3	2.84	2.91
Mist	4	4.01	6.11
CurvedManifold	12	12.73	-
NonLinear6d36d	6	5.96	9.80
Serpinski	1.58	1.61	1.87
Ikeda **Attractor**	1.71	1.00	1.00
FilteredReactgo	unk	7.87	4.70
Fdgo	unk	0.18	0.20
Contextdm1	unk	16.92	84.56

Table 5.2: Computations of 0,1-dim. PHFD for all datasets

results are not so promising. This is a point that needs further analysis in future work.

Finally, for what concerns the estimator $comp_{PH}$ we have tried to replicate the results obtained in [35]. Unfortunately, the results seem to be far from the desiderata. We have some doubts about the global definition of the estimator, since, as shown in the references, the corresponding function F should have a clear linear behavior, in a well-defined interval. In concrete terms, in our application, we are not able to see it. These conclusions, of course, have given us the curiosity to investigate more in depth the definition, and this represents a good direction for future research.

Conclusions

This thesis collects our analysis and results obtained by combining new theoretical techniques about TDA and an already existing software `Smart Prod ACTIVE`, provided by the company of the project *EnginSoft S.p.A.* As explained in SNSI, the main aim of such a collaboration is also to bring about an improvement of productive processes and a corresponding new organization of production. The sofware is a web-based platform that brings companies into the era of data and artificial intelligence, allowing optimization in the production process and many other solutions. Thanks to it, data on a foundry process are available and have been studied with the aim of predicting the quality of a casting at the end of the process. More in detail, the task developed during the company collaboration was solving a classification problem by Support Vector Machine on highly imbalanced datasets. After a detailed analysis of this tool and of foundry data, we have moved to the theory needed for our purposes: the TDA. Thanks to its strong basis in algebraic topology, it is able to extract new and intrinsic information from data, and so it seems a promising choice for our analysis. We have then studied the *TDA Global*, which represents an extension of the KNN method to the structure of simplicial complexes. After underlying and explaining its possible drawbacks, we proposed *TDA Local*, its local variant. Tests have been done to check the performances of such classifiers on general data and then on foundry ones. The results are indeed promising and show how topological methods both, TDA Global and Local, could represent a good alternative to other baseline methods. In this direction, some problems are still open. For example, how to correctly set K in TDA Local and how to improve the performances of topological methods.

Another relevant topic we have analyzed is the classification of Persistent Diagrams using SVM. Of course, thanks to the well-known kernel trick, we need to introduce a suitable definition of kernels that could be evaluated on a couple of PDs. After a deep analysis of the main famous 5 PKs, we have first run some tests to figure out the best values for the parameters. Before our study, such an analysis was completely absent in the literature. Then we were interested in comparing the performance of PKs trying to classify data of different types, such as point clouds, grayscale images, graphs, and signals. The conclusions are indeed clear. SWK ends up being the best choice among all tested kernels mainly for two different reasons: first, it predicts labels better than the other ones, and then it is less expensive from a computational point of view since its preGram matrix doesn't depend on parameters and so can be computed only once at the beginning of the code. It is still an open problem the reason why one the dataset ENZYMES, all kernels meet problems and thus the accuracy is lower that that related to other datasets.

Another interesting application of PH is in the context of *Intrinsic Dimension.* In the Machine Learning framework, the user usually deals with a large amount of data with a huge number of features. So, the number of these essential features is exactly the idea behind the Intrinsic Dimension. First, we outline the most popular methods available today to estimate ID, then we show definitions of ID, such as the box-counting dimension and the correlation dimension, and finally, we point out how PH can be used to create ID estimators, describing **i-Dimensional Persistent Homology Fractal Dimension**, **Persistent Homology Dimension** and **Persitent Homology Complexity**. Our numerical tests reveal how *i-Dimensional Persistent Homology Fractal Dimension* ends up being "optimal" in the sense that, its α generalization obtains its best value when $\alpha = 1$. Then we apply several definitions of ID estimators to data also from the world of Neuroscience to understand how they perform on real data. Probably some further analysis needs to be done to investigate better how the estimator approximates badly the ID of such datasets to improve, in the future, the methods themselves. Concerning $comp_{PH}$, some further analysis is needed with the goal of fixing the bad behavior of the estimator, obtaining a new and good approximation of ID.

For every topic, we have written down our implementations that are available on the corresponding GitHub page, recalled in each chapter.

Appendix A

Some useful materials

A.1 Machine Learning pipeline

In Machine Learning, a common task is the construction of algorithms that can learn from and make predictions on data. Such algorithms work on making data-driven predictions by building a mathematical model from input data. These data are commonly divided into three groups, used in different steps inside the creation process of the model itself: **Training**, **Validation**, and **Test** Sets. The standard procedure consists of splitting the whole dataset into ratios that depend on various factors. Generally, a standard split is 60-80% for Training data, 10-20% for Validation data, and $10 - 20\%$ for Test data. All these sets must have the same probability distribution.

Once you have chosen the model to use, for example, in the context of classification some possible choices are KNN, SVM, NN, DT, the first problem to address is highlighting the best values for the model's parameters.

In fact each model depends on some hyperparameters that need to be tuned accordingly to the data under analysis. This goal is reached through the Validation phase. In the basic approach after fixing a model, one has to define a set of admissible values to test for each hyperparameter. With **Grid Search**, the most conventional algorithm, one tries every combination of the provided values to find out the best set. More precise, the workflow is as follows, see Figure A.1:

1. Divide the dataset into Train, Validation, and Test Sets

2. Define all possible combinations of values of hyperparameters to test

3. For each combination, the corresponding model fits the data of the Training Set

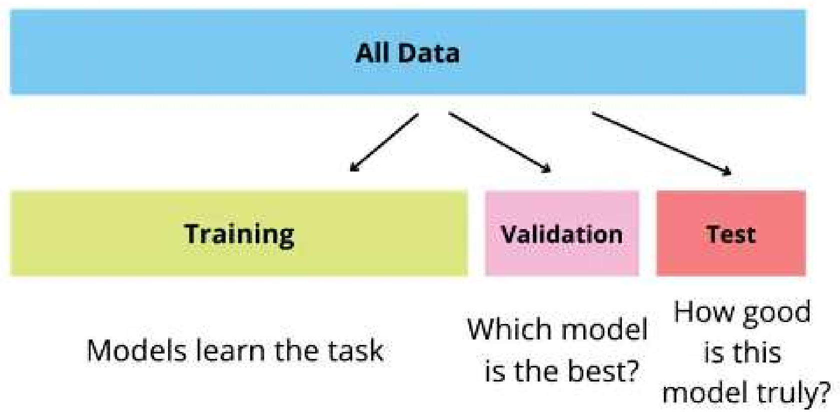


Figure A.1: Example of Machine Learning Workflow

4. Use the model for predicting the label of the Validation Set
5. Measure the quality of prediction of the model, using for example accuracy (number of correct classifications over all samples to classify)
6. Apply the best model to unseen samples of the Test Set

A common approach that reveals more stability in applications is the so-called **Cross Validation (CV)** or better **K-Fold Cross Validation**. Cross-validation is essential for hyperparameter tuning and basically it consists of applying the previous workflow K times to different subdivisions of the initial data into Training and Validation Sets. Figure A.1 shows graphically the idea behind in case with $K = 5$.

The dataset can be divided repeatedly into several Training and Validation Sets and then we need to [74]:

1. Split the data into K folds
2. Train the model on k-1 of the folds (i.e., train the model on all of the folds, except one)
3. Evaluate the model on the kth holdout fold by computing performance on a metric
4. Rotate the folds, and repeat steps 2 and 3, with a new holdout fold. Repeat steps 2 and 3 until all of the k folds have been used as the holdout fold exactly 1 time

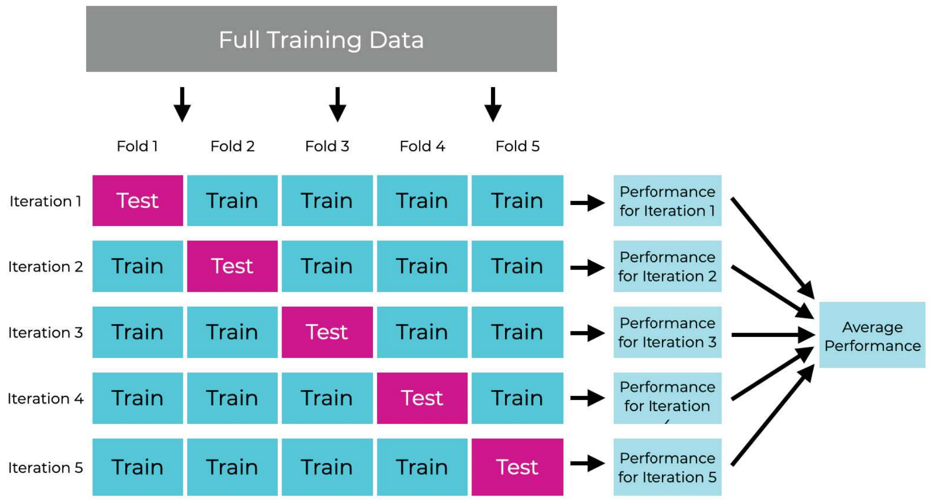

Full Training Data
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Test
Train
Performance for Iteration 1
Performance for Iteration 2
Performance for Iteration 3
Performance for Iteration 4
Performance for Iteration 5
Average Performance

Figure A.2: Example of K-Fold CV with $K = 5$

5. Average the model performance across all iterations

In case of an imbalance dataset, it is better to resort to a suitable variation of K-Fold Cross Validation called Stratified K-Fold Cross Validation.

Stratified Cross Validation is a technique used in Machine Learning to ensure that each fold of the dataset used in cross-validation maintains the same proportion of classes as the entire dataset. This is particularly important in cases where one has imbalanced classes, meaning that some classes have significantly more samples than others. In a typical K-Fold Cross-Validation, the dataset is divided into k subsets (or folds). However, if the classes are imbalanced, some folds might end up with very few or no samples from the minority class, which can lead to misleading performance metrics. Stratified cross-validation addresses this issue by ensuring that each fold is a good representative of the overall class distribution. For example, if you have a dataset with 80% of samples belonging to Class A and 20% to Class B, Stratified Cross Validation will ensure that each fold also has approximately 80% of samples from Class A and 20% from Class B. This way, the model is evaluated more reliably, and one gets a better understanding of its performance across different classes. In application, it indeed represents a great way to improve the robustness of the model evaluation.

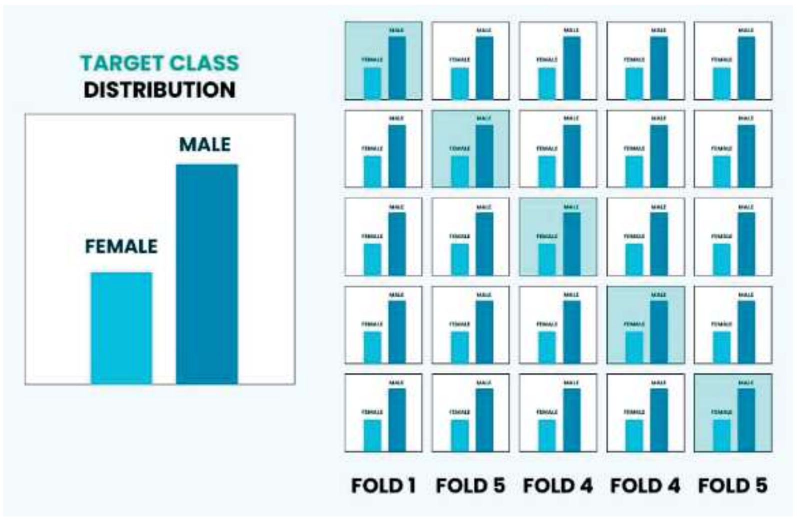


Figure A.3: Example of stratified CV

A.2 Imbalance datasets

Class imbalance is generally normal in classification from real-world scenarios, which usually has only two classes. Some examples of such binary problems are: fraud-, claim-, spam- detection, customer churn prediction, and disease diagnosis that bring severe imbalance datasets. In such contexts, the minority class is called positive, because the model is primarily interested in detecting or predicting it, instead majority class is called the negative one.

In the binary case, once defined and set the model, one ends up with the confusion matrix that collects all information about the classification performances of the model itself, where Actual is the correct and real labels while Predicted collects values assigned by the model used.

In application, usually, the 4 possible groups are denoted through the initials, obtaining TP, FN, FP, and TN, see Figure A.4.

If the dataset is **balanced**, or better if it has equal samples per class, a good measure for prediction quality is represented by *Accuracy*,

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$

that precisely measures the number of correctly classified samples over the total number of points.

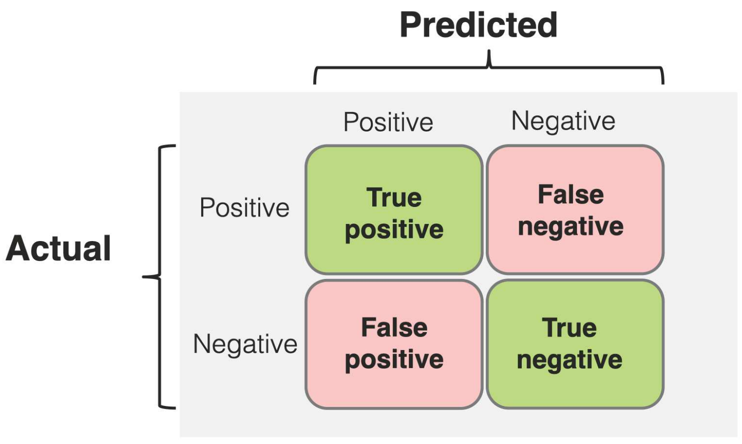


Figure A.4: Confusion Matrix

Instead, it is said to be imbalanced when there is significant, or in some cases extreme disproportion among the number of examples of each class of the problem [27]. The class or classes with abundant examples are called major or majority class, whereas the class with few examples is called minor or minority class.

Definition 40. The **Imbalance Ratio (IR)** is defined as the cardinality of major class (negative class) data instances divided by the number of samples in the minor class (positive class) in binary datasets.

In general, if the dataset is only quite imbalanced, most Machine Learning techniques can face easily the problem. Instead, where the class imbalance is high, e.g. 90% for the majority class and 10% for the minority one as happens frequently in aforementioned contexts, the traditional methods or performance measures may not be so effective and need to be modified. When working with an imbalanced classification problem, the minority class is typically the most interesting one. Therefore one needs a model that can predict correctly samples from minor classes even if such examples are only a few.

An example to clarify the real-world classification problem is outlined by fraudulent transaction detection. It is an important task for credit and companies to prevent huge financial losses. The imbalance comes from the fact that the number of such fraudulent transactions is usually much smaller than that of non-fraudulent ones. In this context, the fraudulent transactions are the ones that we aim to detect. Keeping in mind this example, if one tried to approach the problem as it would be a balanced one, some problems would arise:

1. **Biased Model Training**: due to imbalance, a standard model can achieve high accuracy by simply predicting the majority class ignoring completely the minority one. The resulting model will turn out to be biased towards the majority class and fail to capture samples of the minority class accurately.

2. **Poor Generalization Power**: a model biased toward the majority class has not learned enough about the minority one and so it may struggle to make accurate predictions for instances belonging to that class. This means that the model's power of generalization, for instance, the skill of predicting correctly the belonging class, of new and unseen data is really poor.

3. **Costly Error**: a biased model ends up with misclassification samples that are of two types, the False Positive, and False Negative, see Figure A.4, where the most crucial is the False Negative. To understand better the reason why, let's think about disease diagnosis. In that context, the misclassification in FN is equal to saying that an ill patient is healthy, with serious consequences that are easily understandable.

4. **Evaluation Metric Misleading**: traditional evaluation metrics like accuracy can be misleading in imbalanced datasets. For example, if we have 100 samples, 95 belong to the majority class and 5 to the minor one. If we assume in the extreme case that the classifier predicts all samples to belong to the majority class, the accuracy will be equal to 95%. For balanced datasets, this percentage is indeed good, unfortunately, it hides the 100% misclassification error related to the minority class. Thus dealing with imbalanced datasets requires suitable modifications of standard methods, data, and performance metrics.

To overcome these challenges, various and specialized techniques need to be employed.

A.2.1 Proper selection of evaluation metrics

In case of binary problems, other metrics might be considered as the **Recall**

$$\text{Recall} = \frac{TP}{TP + FN}$$

the **Precision**

$$\text{Precision} = \frac{TP}{TP + FP}$$

and **F1-score**,

$$\text{F1-score} = \frac{2 * \text{ Precision } * \text{ Recall}}{\text{Precision} + \text{Recall}}.$$

Instead, in the multiclass scenario, other metrics have been proposed, for example, **balanced accuracy**. It considers the number of correct predictions per class, or recall, and then takes the average. More precisely, if

$$\text{Recall}_i = \frac{\text{test samples of class i correctly classified}}{\text{all test samples of class i}}$$

then the **Balanced Accuracy**, in case of n classes is,

$$\text{Balanced\_Accuracy} = \frac{\sum_{i=1}^{n} \text{Recall}_i}{n}.$$

A.2.2 Resampling (Oversampling and Undersampling)

The final aim is to have at hand a balanced dataset. Two ways can be followed: on one hand, the minority class can be **oversampled** increasing in such manner the cardinality of the minor class, similarly, one can randomly delete data from the majority class to match them with the minority one. This is the **undersampling** technique.

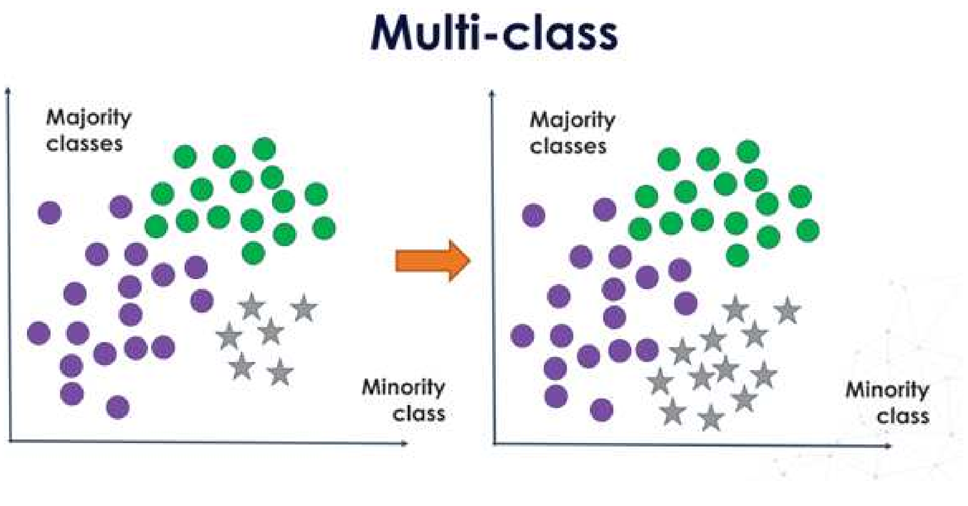
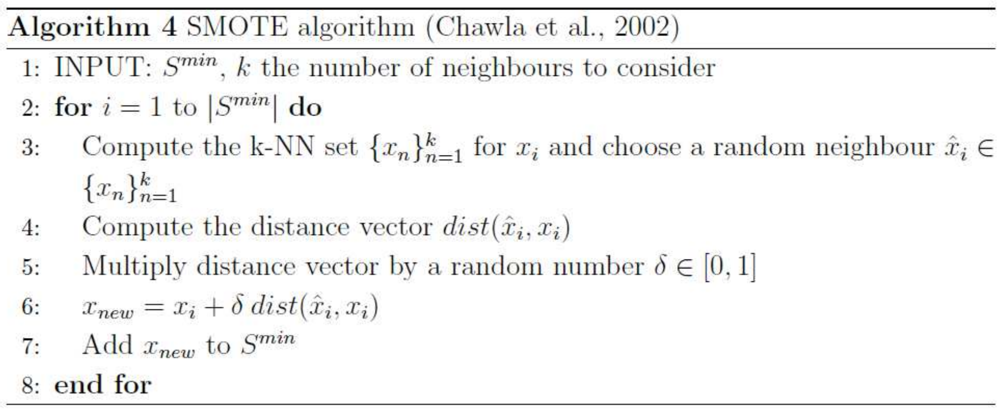


Figure A.5: Example of oversampling of minority class

Choose an over- or undersampling technique, it is not so obvious and depends on the data at hand. Undersampling can be effective when the majority class has many redundant or similar samples or when dealing with huge datasets. However, it can also lead to a loss of information, resulting in biased models. On the other hand, oversampling can be effective when the datasets are small and there are limited available samples of the minority class. But, it can also lead to overfitting due to data duplication or the creation of synthetic data that are not representative of the real data. Thus, in our analysis, we lean towards oversampling. Among oversampling techniques, the most common and used is the **Synthetic Minority Oversampling TEchnique**, or briefly **SMOTE**.

The main idea behind the SMOTE algorithm is to generate synthetic data points of the minority class by interpolating between the minority class instances. In other words, SMOTE creates new data artificially. To achieve this, SMOTE randomly selects a minority class instance and then finds its k nearest minority class neighbors. It then generates new synthetic instances by interpolating between the original minority instance and its k nearest neighbors.



Algorithm 4 SMOTE algorithm (Chawla et al., 2002)

```
1: INPUT: $S^{min}$, $k$ the number of neighbours to consider
2: for $i = 1$ to $|S^{min}|$ do
3:     Compute the k-NN set $\{x_n\}_{n=1}^{k}$ for $x_i$ and choose a random neighbour $\hat{x}_i \in \{x_n\}_{n=1}^{k}$
4:     Compute the distance vector $dist(\hat{x}_i, x_i)$
5:     Multiply distance vector by a random number $\delta \in [0, 1]$
6:     $x_{new} = x_i + \delta \, dist(\hat{x}_i, x_i)$
7:     Add $x_{new}$ to $S^{min}$
8: end for
```

Figure A.6: Scheme of SMOTE

From a computational point of view, the scheme of SMOTE is as in Figure A.6.

SMOTE can generate new samples based on existing ones, which helps to add more information to the dataset to improve model performance. One of the main drawbacks of SMOTE is that it may introduce noise with the synthetic instances, especially when the number of nearest neighbors is set too high. Additionally, SMOTE may not work well on tightly clustered minority class instances or when there are few instances in the minority class.

A.3 SVM for classification

Let $\Omega \subset \mathbb{R}^d$ and $\{\mathbf{x}_1, ..., \mathbf{x}_m\} \subset \mathcal{X} \subset \Omega$ be the set of input data with $d, m \in \mathbb{N}$. We have a training set, composed by the couples $(\mathbf{x}_i, y_i)$ with $i = 1, ..., m$ and $y_i \in \mathcal{Y} = \{-1, 1\}$. The **binary supervised learning task** consists in finding a function $f : \Omega \longrightarrow \mathcal{Y}$, the model, such that it can predict, in a satisfactory way, the label of an unseen $\tilde{\mathbf{x}} \in \Omega \setminus \mathcal{X}$. The binary classification problem can be solved using Support Vectors Algorithms, or rather, it can be seen as an optimization problem, all details have been taken from [58] (Chapter 7). First, we assume that the patterns are **linearly separable**. In most cases, taken from the real world, this is not true and the problem needs to be analyzed in a nonlinear case. We will go in-depth about this in the following section with the introduction of kernels. The idea of the method is that solving the binary pattern recognition problem is equivalent to finding a hyperplane, in a space $\mathcal{H}$ with a dot product that separates the samples based on the two classes. The hyperplane in $\mathcal{H}$ can be written as

$$\{x \in \mathcal{H} : \langle w, \mathbf{x} \rangle + b = 0\}, w \in \mathcal{H}, b \in \mathbb{R}.$$

Obviously, in the equation w, b is not unique since we can multiply the equation for any $c \in \mathbb{R}$ to obtain the same hyperplane. To avoid this possibility from occurring, we introduce the following

Definition 41. The pair $(w, b) \in \mathcal{H} \times \mathbb{R}$ is called a **canonical form** of the hyperplane with respect to $\mathbf{x}_1, \dots, \mathbf{x}_m \in \mathcal{H}$ if it scaled such that

$$\min_{i=1,\dots,m} |\langle w, \mathbf{x}_i \rangle + b| = 1$$

In this case, the closest point to the hyperplane has a distance of $\frac{1}{\|w\|}$.

In this setting, the model that we are looking for will have the following aspect, $f_{w,b} : \mathcal{H} \to \{-1, 1\}$, $\mathbf{x} \mapsto f_{w,b}(\mathbf{x}) = sgn(\langle w, \mathbf{x} \rangle + b)$ and we would like that at least for a large fraction of index i, $f_{w,b}(\mathbf{x}_i) = y_i$.

It can be seen that the larger the margin between the hyperplane and the closest point turns out to be, the higher power of generalizations will be shown by the model. For this reason, these methods are called **Large Margin Algorithms**.

Definition 42. For an hyperplane $\{x \in \mathcal{H} | \langle w, \mathbf{x} \rangle + b = 0\}$, we call

$$\rho_{w,b}(\mathbf{x}, y) := \frac{y(\langle w, \mathbf{x} \rangle + b)}{\|w\|}$$

the **geometrical margin** of the point $(\mathbf{x}, y) \in \mathcal{H} \times \{-1, 1\}$. Instead, the minimum value,

$$\rho_{w,b} := \min_{i=1,\ldots,m} \rho_{w,b}(\mathbf{x}_i, y_i)$$

shall be call the **geometrical margin** of $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$.

Actually, given a generic pattern $\mathbf{x} \in \mathcal{H}$, the geometrical margin is indeed the distance between the point itself and the hyperplane, and the multiplication for y means that the margin is positive. So it is the real distance, if the correct classification takes place, otherwise it is negative.

It turns out that the margin of a separating hyperplane, and thus the length of w, plays a fundamental role in the support vector type algorithm. The main idea behind these methods is that if we manage to separate the training data with a large margin, then probably we will do well on the test set.

Now we focus our attention on the derivation of the optimization problem to compute the **Optimal Margin Hyperplane**. The setting is as before and in addition, we assume that there is at least one positive and one negative y_i.

The problem is finding a decision function, previously introduced,

$$f_{w,b}(\mathbf{x}) = sgn(\langle w, \mathbf{x} \rangle + b) \text{ such that } f_{w,b}(\mathbf{x}_i) = y_i \ \forall i = 1, \ldots, m.$$

If such a function exists, the canonicality introduced before means that the inequalities hold,

$$y_i(\langle w, \mathbf{x}_i \rangle + b) \geqslant 1 \quad \forall i = 1, \ldots, m.$$

So the problem, we want to solve, turns out to be

$$\begin{array}{cl} \min\limits_{w \in \mathcal{H}, b \in \mathbb{R}} & \frac{1}{2}\|w\|^2 \\ \text{s. to} & y_i(\langle w, \mathbf{x}_i \rangle + b) \geqslant 1 \ \forall i = 1, \ldots, m \end{array}$$

It is called the **primal optimization problem**. This is a problem of convex optimization with convex constraints, but, as often done, we introduce the dual problem because that has the same solution as the primal one but in practise is more convenient and easier to treat. With this aim, we introduce the Lagrangian,

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i(y_i(\langle w, \mathbf{x}_i \rangle + b) - 1)$$

with Lagrange multipliers $\alpha_i \geqslant 0$. The Lagrangian L must be maximized w.r.t. α_i and minimized w..r.t. w, b, and then with some observations, we start the formalization of the dual problem. If $S = \mathcal{H} \times \mathbb{R} \times [0, +\infty)^m$ that is

$$\begin{aligned}
\max_{w,b,\alpha \,\in\, S} \quad & L(w, b, \alpha) \\
\text{s. to} \quad & \frac{\partial L(\mathbf{x}, b, \alpha)}{\partial w} = 0 \Leftrightarrow w = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \\
& \frac{\partial L(\mathbf{x}, b, \alpha)}{\partial b} = 0 \Leftrightarrow \sum_{i=1}^{m} \alpha_i y_i = 0 \\
& \alpha_i \geqslant 0 \ \forall i = 1, \ldots, m
\end{aligned}$$

and by replacing the constraints with the Lagrangian, we finally obtain the **dual form** of the optimization problem

$$\begin{aligned}
\max_{\alpha \,\in\, \mathbb{R}^m} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\
\text{s. to} \quad & \sum_{i=1}^{m} \alpha_i y_i = 0 \\
& \alpha_i \geqslant 0 \ \forall i = 1, \ldots, m
\end{aligned} \tag{A.1}$$

Accordingly, with the **Karush Kuhn Tucker (KKT)** necessary condition, we have that in the optimal point the equalities hold

$$\alpha_i [y_i(\langle w, \mathbf{x}_i \rangle + b) - 1] = 0 \ \forall i = 1, \ldots, m.$$

The pattern $\mathbf{x}_i$ for which $\alpha_i > 0$ are called **Support Vectors**. In the first constraint of the first dual formulation, we notice that w can be written as a linear combination of these support vectors and so, once α_i are computed, the w comes easily and is unique. These patterns have to satisfy the constraints and so they lie exactly on the margins, which means also that the hyperplane is in the canonical form.

Thanks to the constraints that arise in the dual problem, the decision functions take the new structure

$$f_{w,b}(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i sgn(\langle \mathbf{x}_i, \mathbf{x} \rangle + b) \tag{A.2}$$

However, it is not always true that the patterns are linearly separable, as shown in the *Cover's theorem* that follows

thm A.3.1. *Let $m \in N$ and $\mathcal{X} = \mathbb{R}^d$ for some $d \in N$. The linear separation of m points in general position in a d-dimensional space is:*

$$2^n \text{ if } m \leqslant d+1$$

$$2\sum_{i=0}^{d}\binom{m-1}{i} \text{ if } m > d+1$$

This result brings us to a more general contest in which through a nonlinear map the data could take place in a higher-dimensional space where instead they could be divided by a hyperplane. With this idea in mind, we analyze the Nonlinear Support Vector Classifiers.

A.3.1 Support Vector Machine with Kernels

In the previous section, we have introduced $\mathcal{X}$ as a subset of some $\mathbb{R}^d$ but indeed the theory holds if we ask only that $\mathcal{X}$ is a set, without any structure. But to be able to generalize, we need some tools that allow us to understand if an unseen pattern $\mathbf{x}$ is "similar" to one in our training pattern. So we have to introduce a function (kernel) κ

$$\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

$$(x, x^{'}) \mapsto k(x, x^{'})$$

that is a function that returns a real number characterizing the similarity between x and $x^{'}$. A particular and simple case is given by the *dot product*,

$$\langle x, x^{'} \rangle = \sum_{i=1}^{m} x_i x_i^{'}.$$

But to do this kind of computation, we need to represent the patterns as vectors in some space $\mathcal{H}$ with a dot product. To this end, we use the map

$$\Phi : \mathcal{X} \to \mathcal{H}$$

$$x \mapsto \mathbf{x}$$

where $\mathbf{x}$ denotes the vector. The space $\mathcal{H}$ is called ***feature space***. This approach can be followed even if $\mathcal{X}$ is already a space with linear product, but one wants to consider a more general similarity obtained by applying the map. Finally, Φ can be also a nonlinear map. Thanks to this function, we might define a similarity measure from the dot product in $\mathcal{H}$ as

$$k(x,x^{'}) := \langle x, x^{'} \rangle = \langle \Phi(x), \Phi(x^{'}) \rangle.$$

Now we can deal with the pattern geometrically and can see the problem from another point of view. Recalling the property of kernels, it follows that the map could be $\Phi(x) = \kappa(\cdot, x)$ with $\mathcal{H}$ the RKHS related to kernel κ. Thus $\mathcal{H}$ is explicitly equal to the space of functions $\mathbb{R}^{\mathcal{X}} = \{f : \mathcal{X} \to \mathbb{R}\}$ and the relation between this one and $\mathcal{X}$ is given by

$$\Phi : \mathcal{X} \to \mathbb{R}^{\mathcal{X}}$$

$$x \mapsto \kappa(\cdot, x).$$

We assume that the kernel κ is positive definite.
Through Φ we might embed patterns into a vector space, called **feature space**

$$\mathcal{F} = \{\sum_{i=1}^{m} \alpha_i \kappa(\cdot, x_i) | n \in N, x_i \in \mathcal{X}, \alpha_i \in R, i = 1, \ldots, m\}.$$

This space is equipped with an inner product defined as

$$f(x) = \sum_{i=1}^{m_1} \alpha_i \kappa(x, x_i), \;\; g(x) = \sum_{j=1}^{m_2} \beta_i \kappa(x, x_j^{'})$$

$$\langle f(x), g(x) \rangle := \sum_{i=1}^{m_1} \sum_{j=1}^{m_2} \alpha_i \beta j \kappa(x_i, x_j^{'})$$

with $f, g \; \in \mathcal{F}$, $x_1, \ldots, x_{m_1}$ and $x_1^{'}, \ldots, x_{m_2}^{'}$ two sets of patterns chosen in $\mathcal{X}$ is a pre-Hilbert space as mentioned in the first section.
Note that using the properties of kernels we obtain that

$$\langle f(x), g(x) \rangle = \sum_{j=1}^{m_2} \beta_j f(x_j^{'}) = \sum_{i=1}^{m_1} \alpha_i g(x_i)$$

thm A.3.2. *A function κ defined on $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a reproducing kernel if and only if there exists a Hilbert space $\mathcal{H}$ and a mapping $\Phi : \mathcal{X} \to \mathcal{H}$, such that for all $x, x^{'} \in \mathcal{X}$*

$$\kappa(x, x^{'}) = \langle \Phi(x), \Phi(x^{'}) \rangle_{\mathcal{H}}$$

This formula states the equivalence between a kernel evaluation and a dot product of feature maps, it is often referred to as **Kernel Trick** in the machine learning literature. This relation is indeed interesting since we don't need to know what is Φ, it is sufficient to compute $\kappa(x, x^{'})$. More precisely about the Kernel Trick, given an algorithm that is formulated in terms of positive definite kernel κ, one can build up an alternative algorithm by replacing κ with another positive definite kernel $\tilde{\kappa}$.

Another interesting result is the continuity of the feature map,

Proposition A.3.3. *If $\mathcal{X}$ is a topological space and κ is a continuous positive definite kernel on $\mathcal{X} \times \mathcal{X}$, then there exists a Hilbert space $\mathcal{H}$ and a continuous map $\Phi : \mathcal{X} \to \mathcal{H}$ such that for all $x, x^{'} \in \mathcal{X}$, we have $\kappa(x, x^{'}) = \langle \Phi(x), \Phi(x^{'}) \rangle$.*

Now we are ready to come back to the classification problem. Thanks to all the previous statements, we replace the scalar product in the previous section with the Kernel κ. For instance, the previous decision function in A.2 becomes

$$f_{w,b}(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i sgn(\kappa(\mathbf{x}_i, \mathbf{x}) + b)$$

and the problem in A.1 can be written as

$$\begin{aligned}
\max_{\alpha \,\in\, \mathbb{R}^m} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \\
\text{s. to} \quad & \sum_{i=1}^{m} \alpha_i y_i = 0 \\
& \alpha_i \geqslant 0 \ \forall i = 1, \ldots, m
\end{aligned}$$

It is easy to see that $W(\alpha)$ is a convex function thanks to the positive definitiveness of the kernel κ and this makes the problem easy to solve efficiently.

Finally using the conditions of **Karush-Kuhn-Tucker (KKT)**,

$$\sum_{i=1}^{m} \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x}_j) + b = y_i$$

and thus

$$b = y_i - \sum_{i=1}^{m} \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x}_j).$$

But in some cases, even if we can transform the data under Φ the linear separation is not possible or it is not always the best solution to the classification problem. In this setting, one can relax the constraints by introducing some appropriate variables and adding a cost in the function to minimize in such a way that this term counts, in a certain way, the number of initial constraints that the solution at the end will not verify. In this way, the algorithm is not even more so rigid and allows the presence of some outliers. This setting is the so-called **soft margin** formulation.

Then, we introduce the **slack variables** ξ_i and with suitable changes the problem becomes

$$\begin{array}{cl} \min\limits_{w \in \mathcal{H}, \xi \in \mathbb{R}^m} & \frac{1}{2}\|w\|^2 + \frac{C}{m}\sum\limits_{i=1}^{m} \xi_i \\ \text{s. to} & y_i(\langle w, \mathbf{x}_i \rangle + b) \geqslant 1 - \xi_i \ \forall i = 1, \dots, m \\ & \xi_i \geqslant 0 \ \ \forall i = 1, \dots, m \end{array}$$

For some $C > 0$, this is the primal form of the soft margin problem, approached with the so-called **C - SV classifier**.

As in the previous case, the solution can be shown to have an expansion

$$w = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i.$$

In the previous case, the coefficients α_i can be found as the solution of the following quadratic programming problem:

$$\begin{array}{cl} \max\limits_{\alpha \ \in \ \mathbb{R}^m} & \sum\limits_{i=1}^{m} \alpha_i - \frac{1}{2}\sum\limits_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \kappa(x_i, x_j) \\ \text{s. to} & \sum\limits_{i=1}^{m} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq \frac{C}{m} \ \forall i = 1, \dots, m \end{array}$$

For this classifier, C represents a hyperparameter that needs to be tuned through CV.

A.4 Decisional Tree for classification

Among Machine Learning techniques, Decision Trees stand out due to their intuitive structure and ease of interpretation. The algorithm is part of the supervised learning algorithms family.

A Decision Tree is a flowchart-like model that represents decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It consists of nodes that represent decisions or tests on features, branches that represent the outcome of these tests, and leaf nodes that represent final decisions or classifications. This hierarchical structure allows for a clear visualization of the decision-making process, making it accessible even to non-experts. On the other hand, its ability to handle both numerical and categorical data makes it versatile and widely applicable. Despite its advantages, such as simplicity and interpretability, Decision Trees also have limitations, including susceptibility to overfitting and sensitivity to small changes in the data.

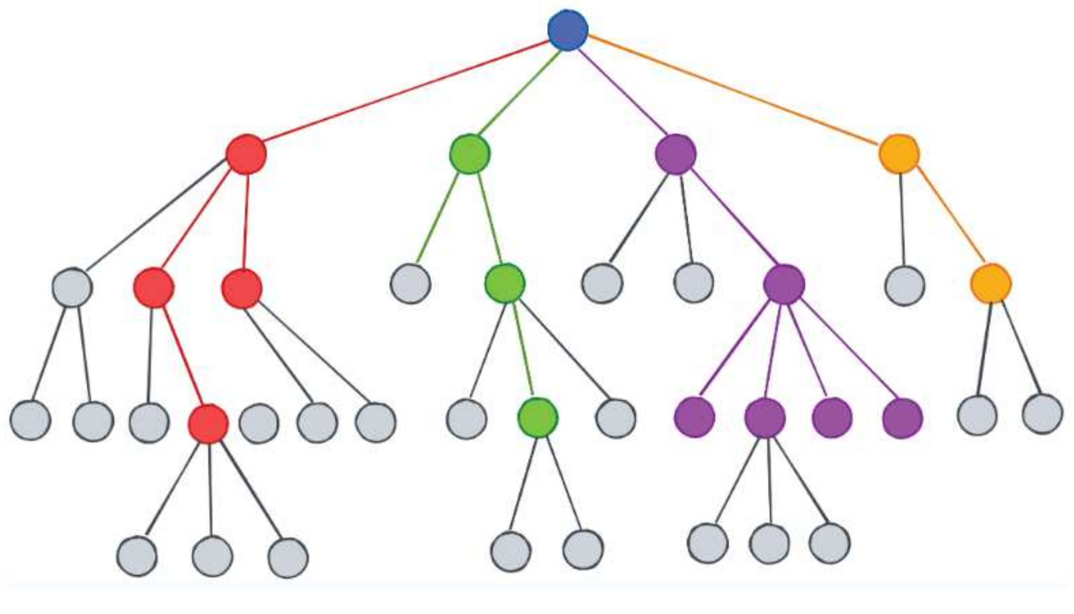


Figure A.7: Example of structure of a Decisional Tree

The structure is similar to that shown in Figure A.7 and the method works following several steps:

1. **Starting Point or Root**: The process begins at the root node, which contains the entire dataset.
2. **Splitting**: The algorithm evaluates the features and splits the data into subsets based on certain criteria. This is done recursively, creating branches for each possible outcome of the decision.
3. **Decision Making**: At each node, a decision is made based on the value of a feature. For example, if you're classifying whether an email is spam or not, a node might check if the email contains certain keywords.
4. **Leaf Nodes**: The process continues until a stopping condition is met, such as reaching a maximum depth or having a minimum number of samples in a node. The final nodes (leaf nodes) represent the predicted class labels.

Some criteria can guide the splitting phase measuring the impurity or disorder of a dataset when building decision trees. They help determine how to split the data at each node in the tree to achieve the best classification results. The most commonly used in the CV phase, as in our tests, are:

- **Gini Index**: This criterion calculates the probability of a randomly chosen element being misclassified if it was randomly labeled according to the distribution of labels in the subset. A lower Gini index indicates a purer node, meaning that most of the samples belong to a single class.
- **Entropy**: This is derived from information theory and measures the amount of uncertainty or disorder in the dataset. It quantifies the expected amount of information needed to classify a sample. Lower entropy values indicate a more homogeneous node.

While they are different in their calculations, they often lead to similar results in practice.

Acknowledgements

The present work represents a journey that started 3 years ago. It has been enriched by a lot of situations, and conferences but among all by people, that I have luckily met. In addition, this work is my personal academic goal, which I was not sure that in my life I wished to reach. Obviously, I think it is honest and important to thank someone in particular, some people who have supported me in reaching this final aim.

First of all, I wish to thank Professor Stefano De Marchi because he convinced me to continue to do research by concluding my ”academic journey”, by writing 3 theses with the same advisor. I also want to thank him because he has allowed me to make relationships with other colleagues all over Italy and abroad.

I wish also to thank two other professors, named also in my research project that are prof. Salvatore Cuomo and prof. Kai Hormann. The first was indeed a good inspiration and made me discover the world of Neural Networks. On the other hand, Prof. Hormann was my supervisor during the 6 months spent in Lugano and he encouraged me to continously feed my curiosity about math.

Professor Roberto Monti, who played an important role in my bachelor's and master's formation, has continued to be my mentor and a point of reference for the PhD period.

Thanks also to Manolo Venturin, my tutor in EnginSoft, for his support and help during the time spent in the company. Another special thanks to Michele Allegra, of the Padova Neuroscience Center, who helped me discover the magic world of Intrinsic Dimension guiding the last part of my research.

A big thanks to all the people I have met in these years: other PhD students and researchers in Padova and other researchers and professors I met at conferences.

I'd like to give special thanks to the people who continue to strongly believe in me and in my skills, especially my parents, who are a concrete fixed point of reference. They trusted me, and even if they were sometimes worried about me, they always gave me strong support.

Then I wish to thank my friends and people who love me and have supported me in these three years.

Finally, I must thank also some hairy friends who made me laugh and enjoy myself during the last year.

Bibliography

[1] Adams H.; Aminian M.; Farnell E.; Kirby M.; Peterson C.; Mirth J.; Neville R.; Shipman P.; Shonkwiler C. A Fractal Dimension for Measures via Persistent Homology. (eds) Topological Data Analysis *Abel Symposia Springer* **2020**, *15*, ISBN: 978-3-030-43407-6

[2] Adams H.; Emerson T.; Kirby M.; Neville R.; Peterson C.; Shipman P.; Chepushtanova S.; Hanson E.; Motta F.; Ziegelmeier L. Persistence images: A stable vector representation of persistent homology. *Journal of Machine Learning Research* **2017**, *18, 1–35*

[3] Aktas M.E.; Akbas E.; El Fatmaoui A. Persistent Homology of Networks: Methods and Applications. *Appl. Netw. Sci.* **2019**, *4, 61*

[4] Ali D.; Asaad A.; Jimenez M.; Nanda V.; Paluzo-Hidalgo E.; Soriano-Trigueros M. A Survey of Vectorization Methods in Topological Data Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **2023**, *45, 14069–14080*

[5] Asaad A.; Ali D.; Majeed T.; Rashid R. Persistent Homology for Breast Tumor Classification Using Mammogram Scans. *Mathematics* **2022**, *10, 21*

[6] Bandiziol C.; De Marchi S. Persistence Symmetric Kernels for Classification: A Comparative Study. *Symmetry* **2024**, *16, 1236* `https://doi.org/10.3390/sym16091236`

[7] Barnes D.; Polanco L.; Peres J.A. A Comparative Study of Machine Learning Methods for Persistence Diagrams. *Frontiers in Artificial Intelligence* **2021**, *4, 28*

[8] Bhattacharya D.; Kaur R.; Aithal N.; Sinha N.; Issac T. G. Persistent homology for MCI classification: a comparative analysis between graph and Vietoris-Rips filtrations. *Front. Neurosci.* **2025**, *19*

[9] Brüel-Gabrielsson R.; Ganapathi-Subramanian V.; Skraba P.; Guibas L.J. Topology-Aware Surface Reconstruction for Point Clouds. *Computer Graphics Forum* **2020**, *39, 197–207*

[10] Bubenik P. Statistical Topological Data Analysis using Persistence Landscapes. *Journal of Machine Learning Research* **2015**, 16, 77-102

[11] Bukkuri A.; Andor N.; Darcy I. K. Applications of Topological Data Analysis in Oncology. *Front. Artif. Intell.* **2021**, 4, 659037

[12] Camastra F.; Staiano A. Intrinsic dimension estimation: Advances and open problems. *Information Sciences* **2016**, *328, 26–41*

[13] Campadelli P.; Casiraghi E.; Ceruti C.; Rozza A. Intrinsic Dimension Estimation: Relevant Techniques and a Benchmark Framework. *Information Analysis of High-Dimensional Data and Applications* **2015**

[14] Carlsson G. Topology and data. *Bulletin of the American Mathematical Society* **2009**, *46, 255–308*

[15] Carlsson G.; Vejdemo-Johansson M. Topological Data Analysis with Applications. *Publisher: Cambridge University Press*, **2022**, ISBN: 978-110-883-865-8

[16] Carrière M.; Cuturi M.; Oudot S. Sliced Wasserstein kernel for persistent diagrams. *International Conference on Machine Learning*, **2017**, *664–673*

[17] Carrière M.; Chazal F.; Ike Y.; Lacombe T.; Royer M.; Umeda Y. PersLay: A Neural Network Layer for Persistence Diagrams and New Graph Topological Signatures. *International Conference on Artificial Intelligence and Statistics* **2019**

[18] Chazal F.; Michel B. An introduction to Topological Data Analysis: Fundamental and Practical Aspects for Data Scientists. *Front. Artif. Intell.* **2021**, *4*

[19] Chung Y.-M.; Cruse W.; Lawson A. A Persistent Homology Approach to Time Series Classification. *ArXiv* **2020**, *submitted*

[20] Cohen-Steiner D.; Edelsbrunner H.; Harer J. Stability of persistence diagrams. *Discrete & computational geometry* **2007**, *37, 103–120*

[21] Dau H.A.; Keogh E.; Kamgar K.; Yeh C.M.; Zhu Y.; Gharghabi S.; Ratanamahatana C.A.; Yanping; Hu B.; Begum N.; Bagnall A.Y.; Mueen A.; Batista G.; *Hexagon-ML. University of California Riverside.* Available online: https://www.cs.ucr.edu/ eamonn/time series data 2018/ (accessed on October **2018**)

[22] De Marchi S.; Lot F.; Marchetti F. Kernel-based methods for persistent homology and their applications to Alzheimer's Disease. *Master Thesis, University of Padova*, Padova, 25 June **2021**

[23] De Marchi S.; Lot F.; Marchetti F.; Poggiali D. Variably Scaled Persistence Kernels (VSPKs) for persistent homology applications. *Journal of Computational Mathematics and Data Science* **2022**, *4, 100050*

[24] Edelsbrunner H.; Harer J. Persistent homology - a survey. *Contemporary Mathematics* **2008**, *453, 257–282*

[25] Edelsbrunner H.; Harer J. Computational topology: An introduction. *Publisher: American Mathematical Society*, **2010**

[26] Fasshauer G.E. Meshfree approximation with MATLAB. *Publisher: World scientific*, Singapore, **2007**, ISBN: 978-981-270-634-8

[27] Fernández A.; García S.; Galar M.; Prati R.C.; Krawczyk B.; Herrera F. Learning from Imbalanced Data Sets. *Publisher: Springer Nature Switzerland*, **2018**, ISBN: 978-3-319-98073-7

[28] Flammer M. Persistent Homology-Based Classification of Chaotic Multi-variate Time Series: Application to Electroencephalograms. *SN COMPUTER SCIENCE* **2024**, *5, 107*

[29] Fomenko A.T. Visual geometry and topology *Publisher: Springer Science and Business Media*, **2012**

[30] Garin A.; Tauzin G. A Topological "Reading" Lesson: Classification of MNIST using TDA. *In 18th IEEE International Conference On Machine Learning And Applications*, Boca Raton, FL, USA, **2019**, *1551-1556*

[31] Grandini M.; Bagli E.; Visani G. Metrics for multi-class classification: an overview. *ArXiv* **2020**, *submitted*

[32] Guillemard M.; Iske A. Interactions between kernels, frames and persistent homology. *Recent Applications of Harmonic Analysis to Function Spaces, Differential Equations, and Data Science, Springer* **2017**, *861-888*

[33] Hajij M.; Zamzmi G.; Batayneh F. TDA-Net: Fusion of Persistent Homology and Deep Learning Features for COVID-19 Detection From Chest X-Ray Images. *43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)* **2021**, 4115-4119

[34] Hensel F.; Moor M.; Rieck B. A Survey of Topological Machine Learning Methods. *Front. Artif. Intell.* **2021**, 4

[35] Jaquette J.; Schweinhart B. Fractal dimension estimation with persistent homology: A comparative study. *Communications in Nonlinear Science and Numerical Simulation* **2020**, *84, 105163*

[36] Karan A.; Kaygun A. Time series classification via topological data analysis. *Expert Systems with Applications*, **2021**, *183, 115326*

[37] Kim K.; Kim J.; Zaheer M.; Kim J. S.; Chazal F.; Wasserman L. PLLay: Efficient Topological Layer based on Persistent Landscapes. *Neural Information Processing Systems* **2020**

[38] Kindelan R.; Frías J.; Cerda M.; Hitschfeld N. A topological data analysis based classifier. *Advances in Data Analysis and Classification* **2024**, Issue 2/2024

[39] Kusano G.; Fukumizu K.; Hiraoka Y. Kernel method for persistence diagrams via kernel embedding and weight factor. *The Journal of Machine Learning Research* **2017**, *18, 6947-6987*

[40] Lawson A.; Chung Y.-M.; Cruse W. A Hybrid Metric based on Persistent Homology and its Application to Signal Classification. *2020 25th International Conference on Pattern Recognition (ICPR)* **2020**

[41] Le T.; Yamada M. Persistence fisher kernel: A riemannian manifold kernel for persistence diagrams. *In 32nd Conference on Neural Information Processing Systems*, Montréal, Canada, **2018**

[42] LeCun Y.; Cortes C. MNIST handwritten digit database. Available online: https://yann.lecun.com/exdb/mnist/ **2010**

[43] Lee D.; Lee S.H.; Jung J.H. The effects of topological features on convolutional neural networks—an explanatory analysis via Grad-CAM. *Machine Learning: Science and Technology* **2023**, *4, 21*

[44] Leykam D.; Angelakis D. G. Topological data analysis and machine learning. *Advances in Physics* **2023**, 8(1)

[45] Majumdar S.; Laha A.K. Clustering and classification of time series using topological data analysis with applications to finance. *Expert Systems with Applications* **2020**, *162, 113868*

[46] Moon C.; Li Q.; Xiao G. Using persistent homology topological features to characterize medical images: Case studies on lung and brain cancers. *Ann. Appl. Stat.* **2023**, 17

[47] Moroni D.; Pascali M.A. Learning topology: bridging computational topology and machine learning. *Pattern Recognition and Image Analysis* **2021**, *31, 443-453*

[48] Pachauri D.; Hinrichs C.; Chung M.K.; Johnson S.C.; Singh V. Topology based Kernels with Application to Inference Problems in Alzheimer's disease. *IEEE Transactions on Medical Imaging* **2011**, 30, 1760–1770

[49] Pedregosa F.; Varoquaux G.; Gramfort A.; Michel V.; Thirion B.; Grisel O.; Blondel M.; Prettenhofer P.; Weiss R.; Dubourg V.; Vanderplas J.; Passos A.; Cournapeau D.; Brucher M.; Perrot M.; Duchesnay E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **2011**, *12, 2825-2830*

[50] Pereira A. M. M.; De Mello R. F. Persistent homology for time series and spatial clustering. *Expert Systems with Applications* **2015**, 42, 6026-6038

[51] Phillips J. M.; Venkatasubramanian S. A Gentle* Introduction to the Kernel Distance. *ArXiv* **2011**

[52] Pickup D.; Sun X.; Rosin P.L.; Martin R. R.; Cheng Z.; Lian Z.; Aono M.; Ben Hamza A.; Bronstein A.; Bronstein M.; Bu S.; Castellani U.; Cheng S.; Garro V.; Giachetti A.; Godil A.; Han J.; Johan H.; Lai L.; Li B.; Li C.; Li H.; Litman R.; Liu X.; Liu Z.; Lu Y.; Tatsuma A.; Ye J. SHREC' 14 Track: Shape Retrieval of Non-Rigid 3D Human Models. *Proceedings of the 7th Eurographics workshop on 3D Object Retrieval, EG 3DOR'14. Eurographics Association* **2014**

[53] Pun. C. S.; Lee S. X.; Xia K. Persistent-homology-based machine learning: a survey and a comparative study. *Artif Intell Rev* **2022** 55, 5169–5213

[54] Ravinshanker N.; Chen R. An introduction to persistent homology for time series. *WIREs Computational Statistics* **2021**, *13, e1548*

[55] Reininghaus J.; Huber S.; Bauer U.; Kwitt R. A Stable Multi-Scale Kernel for Topological Machine Learning. *Proceedings of the IEEE conference on computer vision and pattern recognition* **2015**, *4741–4748*

[56] Rotman J. J. *An introduction to Algebraic Topology*; *Publisher: Springer* **1988**

[57] Saul N.; Tralie C. Scikit-tda: Topological data analysis for Python. Available online: https://doi.org/10.5281/zenodo.2533369 (**2019**)

[58] Scholkopf B.; Smola A.J. Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond. *The MIT Press* **2002**, ISBN: 978-026-225-693-3

[59] Shawe-Taylor J.; Cristianini N. Kernel Methods for Pattern Analysis. *Publisher: Cambridge University Press*, **2009**, ISBN: 978-051-180-968-2

[60] Skaf Y.; Laubenbacher R. Topological data analysis in biomedicine: A review. *Journal of Biomedical Informatics* **2022**, 130, 104082

[61] Som A.; Choi H.; Ramamurthy K. N.; Buman M. P.; Turaga P. PI-Net: A Deep Learning Approach to Extract Topological Persistence Images. *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* **2020**, 3639-3648

[62] Sonego P.; Pacurar M.; Dhir S.; Kertész-Farkas A.; Kocsor A.; Gáspári Z.; Leunissen J.A.M.; Pongor S. A Protein Classification Benchmark collection for machine learning. *Nucleic Acids Research* **2006**

[63] Sun J.; Ovsjanikov M.; Guibas L. A Coincise and Provably Informative Multi-Scale Signature Based on Heat Diffusion. *Computer Graphics forum* **2009**, *28, 1383–1392*

[64] Tralie C.; Saul N.; Bar-On R. Ripser.py: A lean persistent homology library for Python. *The Journal of Open Source Software* **2018**

[65] Townsend J.; Micucci C.P.; Hymel J. H.; Maroulas V.; Vogiatzis K. D. Representation of molecular structures with persistent homology for machine learning applications in chemistry. *Nat. Commun* **2020**, *11, 3230*

[66] Xiao H.; Rasul K.; Vollgraf R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *ArXiv* 2017

[67] Yang G.R.; Joglekar M.R.; Song H.F.; Newsome W.T.; Wang X.J. Task representations in neural networks trained to perform many cognitive tasks *Nature neuroscience* **2019**, *22 (2), 297-306*

[68] Zhao Q.; Wang Y. Learning metrics for persistence-based summaries and applications for graph classification. *Proceedings of the 33rd International Conference on Neural Information Processing Systems* **2019**, *884, 9859-9870*

[69] Zomorodian A.; Carlsson G. Computing Persistent Homology. *Discrete Comput. Geom.* **2005** *33, 249–274*

[70] The GUDHI Project, GUDHI User and Reference Manual, 3.5.0 Edition, GUDHI Editorial Board. Available online: https://gudhi.inria.fr/doc/3.5.0/ (**2022**)

[71] Giotto-tda 0.5.1 documentation. Available online: https://giotto-ai.github.io/gtda-docs/0.5.1/library.html (**2021**)

[72] `https://medium.com/@sachinsoni600517/k-nearest-neighbours -introduction-to-machine-learning-algorithms-9dbc9d9fb3b2`

[73] `https://www.researchgate.net/publication/357285351_Quantum_ K-nearest_neighbor_classification_algorithm_based_on_Hamming_ distance/figures?lo=1`

[74] `https://www.sharpsightlabs.com/blog/cross-validation-explained/`

[75] Available online `https://mrzv.org/software/dionysus2/`

[76] Available online `https://github.com/DIPHA/dipha?tab=readme-ov-file`

[77] `https://imbalanced-learn.org/stable/`