UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques
i Informàtica

DOBLE GRAU DE MATEMÀTIQUES I ENGINYERIA INFORMÀTICA

Treball final de grau

TOPOLOGICAL DATA ANALYSIS APPLIED TO MULTIMODAL RECOMMENDATION

Autora: Esther Ruano Hortoneda

Directora Informàtica:	**Dra. Maria Salamó Llorente**
Directora Matemàtiques:	**Aina Ferrà Marcús**
Realitzat a:	**Departament de Matemàtiques i Informàtica**

Barcelona, **17 de gener de 2024**

Contents

Introduction **viii**

1 Introduction **1**
1.1 Project scope . . . 1
1.2 Problem description . . . 2
1.3 Project objectives . . . 2
1.4 Project planning . . . 3
1.5 Report . . . 3

2 Related work **5**
2.1 Topological Data Analysis . . . 5
2.1.1 Homology . . . 5
2.1.2 Filtrations and filtering functions . . . 11
2.1.3 Persistent homology . . . 14
2.1.4 Filtering function stability . . . 21
2.2 Recommender Systems . . . 25
2.2.1 Foundations of Recommender Systems and representation . 26
2.2.2 Recommendation models . . . 28

3 Analysis, design and implementation **35**
3.1 Model choice . . . 35
3.2 Code organization . . . 36
3.3 LATTICE with TDA . . . 37
3.3.1 Extraction of visual features . . . 38
3.3.2 Extraction of text features . . . 39
3.3.3 Item graph . . . 40

4 Experiments **41**
4.1 Research questions . . . 41
4.2 Experimental settings . . . 41

4.3 Evaluation protocol 43
4.4 Model comparison 44
4.4.1 LATTICE 45
4.5 LATTICE updated with TDA 45
4.5.1 LATTICE with TDA in "A" 46
4.5.2 LATTICE with TDA in "B" 47

5 Conclusions and future work **49**

Bibliography **51**

Result tables **55**
.1 Recommender Systems 55
.2 LATTICE with TDA in "A" 55
.3 LATTICE with TDA in "B" 55

List of Tables

4.1 **Dataset description:** statistics of the datasets used to evaluate the Recommender Systems. 42

4.2 **LATTICE fine-tuning:** optimal values for LATTICE in each dataset . 45

4.3 **Model comparison.** Overall performance comparison of LATTICE with different baselines, in terms of Recall (R@20), Precission (P@20) and NDCG@20. We mark with **bold** the best model for each dataset, and with *italic* the second one. 46

4.4 **Model comparison.** Overall performance comparison of LATTICE against different modifications , in terms of Recall (R@20), Precission (P@20) and NDCG@20. We mark with **bold** improvements of over 2.5%, and with *italic* improvements less than 2% and over 1.5% and underline other improvements 48

1 **Matrix factorization:** performance comparison of the Recommender System MF with different datasets. 56

2 **NGCF:** performance comparison of the Recommender System NGCF with different datasets. 56

3 **LightGCN:** performance comparison of the Recommender System MF with different datasets. 56

4 **GRCN:** performance comparison of the Recommender System GRCN with different datasets. 57

5 **MMGCN:** performance comparison of the Recommender System MMGCN with different datasets. 57

6 **VBPR:** performance comparison of the Recommender System VBPR with different datasets. 57

7 **LATTICE:** performance comparison of the Recommender System LATTICE with different datasets. 58

8 **LATTICE with TDA image preprocessing:** performance of LATTICE with TDA with image preprocessing in different datasets. The LATTICE baseline is provided in the last line to facilitate ease of comparison. 58
9 **LATTICE with TDA text preprocessing:** performance of LATTICE with TDA with text preprocessing in different datasets. The LATTICE baseline is provided in the last line to facilitate ease of comparison. 59
10 **LATTICE with TDA image and text preprocessing:** performance of LATTICE with TDA with image and text preprocessing in different datasets. The LATTICE baseline is provided in the last line to facilitate ease of comparison. 59
11 **LATTICE with TDA in the original graph:** performance of LATTICE with TDA on the original graph in different datasets. The LATTICE baseline is provided in the last line to facilitate ease of comparison. 60
12 **LATTICE with TDA on each graph:** performance of LATTICE with TDA on each original graph in different datasets. The LATTICE baseline is provided in the last line to facilitate ease of comparison. . 61
13 **LATTICE with node dropping:** performance of LATTICE with node dropping in the Musical Instruments dataset. The LATTICE baseline is provided in the last line to facilitate ease of comparison. . . . 61
14 **LATTICE with node dropping:** performance of LATTICE with node dropping in the Digital Music dataset. The LATTICE baseline is provided in the last line to facilitate ease of comparison. 62
15 **LATTICE with node dropping:** performance of LATTICE with node dropping in the Baby dataset. The LATTICE baseline is provided in the last line to facilitate ease of comparison. 62

List of Figures

1.1 **Gantt diagram.** Reflects the temporal planning of the tasks required to complete this project. 4

2.1 **Examples of n-simplices.** From left to right: 0-simplex, 1-simplex, 2-simplex, 3-simplex. 6

2.2 **Ordered standard 2-simplex (Δ^2).** Unit points of the coordinate axes are vertices, and the edges have an orientation according to increasing subindices. 7

2.3 **Boundaries of simplices.** From top to bottom, the boundary of a 1-simplex, 2-simplex and 3-simplex. It uses signs so that all the faces of the simplex are oriented. Signs in the formulas are $+1$ if the orientation of a face coincides with the orientation induced by the simplex on that face, and -1 otherwise. 8

2.4 **Filtered cubical complex.** Each vertex, edge and cube has a value assigned by a function f_v, f_e and f_c respectively. 10

2.5 **Freudenthal triangulations.** From left to right: 2D cubical complex, its triangulation; 3D cubical complex, its triangulation. This technique can be used to compute homology for cubical complexes, but it is costly, since the amount of simplices to consider is much greater than the number of cubes. 10

2.6 **Cubical complex homology**. Three cubical complexes with their connected components and cycles listed below using a T-construction. 11

2.7 **Vietoris-Rips filtration of a weighted undirected graph.** For each value of r, we find, first, 4 connected components that later, for $r = 3$, merge into two. For $r = 4$ we find a cycle that disappears when $r = 5$, because it gets filled with simplices. 13

2.8 **Flagser filtration.** In this picture, two triangular directed graphs are shown. The first one is directed acording to increasing subindices, and this makes it a 2-simplex according to the Flagser filtration. The second graph, instead, is not correctly oriented to be an ordered 2-simplex and this is why it is categorized as a hole, according to the Flagser filtration. 13

2.9 **Cubical filtration.** For each value of r we find, first, two connected components, that later merge and at $r = 3$ form a cycle. Lastly, it fills up into a connected component. 14

2.10 **Example graph.** We compute the topological descriptors of this weighted unoriented graph as we define them. 15

2.11 **Persistence barcode and persistence diagram of the example graph, using a Vietoris-Rips filtration.** We consider an increasing radius and, for each value, count how many connected components and cycles are in the graph. These values are monitored on a persistence barcode, and we separate them depending on which homology group they belong: H_0 for connected components and H_1 for cycles. On the persistence diagram, each point (b_i, d_i) represents the death and birth of one of the homology generators. 16

2.12 **Betti curve of the example graph.** It shows how many homology generators are alive at each moment. We have computed a Betti curve for generators in H_0 (straight line) and another one for generators in H_1 (dotted line). 20

2.13 **Tent function**. $\Lambda_{(b,d)}(t) = \max\{0, \min\{\ t - b, d - t\}\}$. 20

2.14 **Landscapes of the example graph.** On the left, the persistence diagram for the generators in H_0. On the right, the landscapes for the first seven generators. We see that, since the multiset is $\{(10,1),(7,3),(5,3)\}$, the $kmax$ function returns 0 for every $k > 7$. Therefore, all the k-landscapes with $k > 7$ are the constant function $\lambda_k = 0$. 20

2.15 **Silhouettes of the example graph:** Silhouette with constant weight (left) and quadratic weight (right). 21

2.16 **Cubical filtration descriptors.** On this image we portray the cubical filtration for a weighted cubical complex, and compute its barcode and persistence diagram. From that, we extract the corresponding Betti curve and the first landscape for both H_0 and H_1. It is important to realise that, since there is only one homology generator for each category that has not infinite persistence, all landscapes λ_k with $k > 1$ are the constant function $\lambda_k = 0$. Lastly, since both points have persistence 1, their silhouettes coincide for $w_i = 1$ and $w_i = (d_i - b_i)^2$. 22

3.1 LATTICE workflow 38

Abstract

This project incorporates Topological Data Analysis, TDA, into a multimodal collaborative Recommender System. To do so, it conducts a comprehensive investigation into the tools that topology provides us, and how they can be used to describe data. Then, they are incorporated into a state-of-the-art multimodal Recommender System in many ways: (i) collecting data about product images and textual descriptions of products to enhance the information that the Recommender System receives, and (ii) extracting information about the item graph and utilizing it both to feed the neural network and to prune the graph, aiming to increase speed without losing performance.

The analysis has been performed on three well-known datasets and with different collaborative models. The results vary depending on the characteristics of the data studied. Generally speaking, we have found that with smaller datasets the performance increases. We have also seen that the information extracted from images seems to be more useful than the information derived from text descriptions. The changes in the network architecture have not been fruitful, and the run time reduction had major impact on the result.

In summary, we observe that the topological data is useful, and we see an improvement in the performance on specific steps of the implementation and with smaller datasets. It can be used in many ways, and this requires a careful preliminary study to evaluate where it can have the best and most meaningful impact.

Resum

En aquest projecte incorporem l'Anàlisi de Dades Topològica, TDA, en un sistema de recomanació multimodal. Per fer-ho, duem a terme una profunda investigació en les eines que la topologia ens facilita, i com les podem fer servir per descriure dades. Després, les incorporem en un sistema de recomanació multimodal modern de diverses maneres: (i) recollint dades sobre imatges i descripcions textuals de productes, per enriquir la informació que el sistema de recomanació rep i (ii) extraient informació sobre el graf que relaciona productes tant per transmetre-la dins la xarxa neuronal com per retallar el graf, aspirant a reduir el temps d'execució sense sacrificar resultats.

Hem dut a terme aquesta anàlisi sobre tres *datasets* coneguts i amb diversos models col·laboratius. Els resultats canvien en funció de les característiques de les dades estudiades. En general, hem vist que els resultats milloren amb *datasets* petits. També hem vist que la informació extreta d'imatges sembla ser més útil

2020 Mathematics Subject Classification. 18G85, 54H30, 55N31, 55U10, 55U15, 62R40, 68U15

que l'extreta de descripcions textuals. Els canvis dins l'arquitectura de la xarxa neuronal no han estat fructífers i, quan hem reduït el temps d'execució, hi ha hagut un gran impacte sobre els resultats.

En resum, hem observat que la informació topològica és útil, i veiem una millora en els resultats en passos específics de la implementació, sobretot en *datasets* petits. Pot ser utilitzada de diverses maneres, i requereix un estudi previ per entendre on pot tenir un impacte més significatiu.

Resumen

En este proyecto incorporamos el Análisis de Datos Topológico, TDA, en un sistema de recomendación multimodal. Para hacerlo, investigamos qué herramientas nos facilita la topología, y cómo podemos usarlas para describir datos. Después, las incorporamos a un sistema de recomendación multimodal moderno de diversas formas: (i) recogiendo datos sobre imágenes y descripciones textuales de productos, para enriquecer la información que el sistema de recomendación recibe y (ii) extrayendo información sobre el grafo que relaciona productos, tanto para después transmitirla dentro de la red neuronal como para podar el grafo, y aspiramos a reducir el tiempo de ejecución sin sacrificar resultados.

Hemos realizado este análisis sobre tres *datasets* famosos y con diversos modelos colaborativos. Los resultados cambian en función de las características de los datos estudiados. En general, hemos visto que los resultados mejoran con *datasets* pequeños. También hemos visto que la información extraída de imágenes parece ser más útil que la extraída de descriptores textuales. Los cambios hechos en la arquitectura de la red neuronal no han sido fructíferos y, cuando hemos reducido el tiempo de ejecución, ha habido un gran impacto sobre los resultados.

En resumen, hemos observado que la información topológica es útil, y vemos una mejora en los resultados en pasos específicos de la implementación, sobre todo para *datasets* pequeños. La topología puede ser usada de diversas formas, y requiere un estudio previo para entender dónde puede tener un impacto más significativo.

Chapter 1

Introduction

This first chapter aims to introduce the areas of research related to this work and set the narrative line that is followed. First, we cover the project scope and set the methods used. Later, we state the problem description and our main goal, and break it down into smaller tasks. To finish up with the introduction, we plan this tasks in the time that we have available for this project and set the structure for this assignment. Before starting, we want to note that all the illustrations included in this report have been made by the author, Esther Ruano.

1.1 Project scope

In this project, we use methods from two different research areas and merge them together: Algebraic Topology and Recommender Systems. First, we look into the area of Algebraic Topology, that is a field in mathematics that combines techniques from abstract algebra and tools coming from topology to study topological spaces [9]. It focuses on algebraic invariants that can be associated with topological spaces and how those are invariant against some types of transformations or deformations. In particular, we introduce Topological Data Analysis (TDA), that focuses on studying different types of data, such as graphs or images.

The second research area that we use are Recommender Systems (RS). Those are a type of algorithms that are designed to suggest items or content to users based on their preferences, behavior, or historical data [18]. These systems are widely used in various online platforms to enhance user experience by providing personalized recommendations and obtaining user engagement. The data used in RSs may come in different ways: text, images, video, audio, etc. The multimodal nature of the data makes it difficult to obtain good descriptors for it, and RSs are usually classified depending on which data they use and how they use it.

This project first looks into Algebraic Topology theory, and investigates how

the methods provided can be used to compute descriptors for many different data types. Afterwards, it studies Recommender Systems, focusing in collaborative systems and diving deep into LATTICE, which uses multimodal features.

Then, with a profound study of the workflow in this particular Recommender System, it highlights points that could be analyzed topologically and it does so. With the information extracted, it adapts the model to incorporate the information obtained in the previous step and, finally, it evaluates the changes made.

1.2 Problem description

The problem that we focus on is obtaining better descriptors for the data. We address it with topological descriptors, extracted from item data, to improve the recommendations our system makes by enhancing the accuracy of our suggestions and aligning them, as much as possible, with the user's preferences. With this, we aim to see if the geometric properties of both image and text descriptors, as well as the ones from a graph that plots item relationships, can help us make more spot-on suggestions to our users and enhance user experience and user satisfaction.

Since there is a large set of RS algorithms in the literature, one important decision to make is choosing which Recommender System to modify. In this work, we have opted for a state-of-the-art multimodal collaborative system, called LATTICE [32] and published in 2021, since it allows both (i) to obtain great results and, (ii) it also admits topological descriptors computation in various ways.

1.3 Project objectives

In this section we describe our main goal, and how we have separated it into smaller tasks to help us plan the work that needs to be done to accomplish it.

The main goal is to understand the methods that Algebraic Topology facilitates to understand data and use it to proportionate descriptors that a RS uses, with the aim of making suggestions that are more adequate to the users preferences.

This implies, first of all, understanding the fundaments of this mathematical branch, and having some notions on what homology and filtrations are. Moreover, investigating which Recommender Systems are available to us, and finding one that provides the best results possible, but that at the same time may have an architecture suitable for computing topological descriptors. Lastly, it is important to be able to benchmark these results against each other, and pick whichever moves us closer to the user preferences. To pursue this main goal; we have set a list of six objectives, that are be detailed in the following chapters:

1. Understand the methods used for Topological Data Analysis.
2. Prove that topological descriptors are robust against structural changes.
3. Study different RSs that deal with multimodal sources of data.
4. Find a RS that can benefit from TDA, and analyze how to incorporate it.
5. Modify the selected RS multimodal model to incorporate TDA.
6. Compare the different RSs against each other and modifications.

1.4 Project planning

To accomplish the objectives listed before, we have divided the project into tasks, and structured them into three phases. To keep this project organized, we have planned each of the tasks along the 17 weeks that go from 15 of September to 15 of January. We structured it into the Gantt diagram, that can be seen in Fig. 1.1.

The first phase involves investigating both the mathematical methods necessary for topology computations, as well as the most adequate algorithms for our recommendation goal. The second phase consists in writing the code for computing topological descriptors on many types of data, and incorporating it into the chosen recommender: LATTICE. Lastly, we sum up all the information retrieved on the first step and the results obtained on the second, and write a report.

1.5 Report

To structure this report, we have followed the same schema used on the project planning: first we introduce all the theoretical concepts used, starting with Algebraic Topology and finishing in Recommender Systems. Then, we implement the code, and analyze the results. Finally, we conclude with the lessons learned and future work. This structure is reflected in the chapters this project is organized in:

- **Chapter 1: Introduction.** Frames the research areas involved, and describes the problem approached. Lays out the project goals and the methods used to pursue them, along with the timeline followed to reach those objectives.
- **Chapter 2: Related work.** Looks at the methods topology provides, and focuses on the ones that are used later. Then, it introduces Recommender Systems and describes seven of them, from the state-of-the-art approaches that deal with multimodal RSs, with special attention to LATTICE and its architecture, since it is the one that we modified to include TDA.

Month	Sept.		Oct.				Nov.					Dec.				Jan.	
Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Phase 1: Investigation																	
1.1 What is TDA																	
1.2 Algebraic topology																	
1.3 Homology																	
1.4 Recommenders																	
1.5 LATTICE code																	
Phase 2: Code implementation and experiments																	
2.1 Adapt MF, NGCF, LightGCN																	
2.2 Adapt GRCN, MMGCN, VBPR																	
2.3 Compute descriptors for images																	
2.4 Compute descriptors for directed graph																	
2.5 Compute descriptors for undirected graph																	
2.6 Incorporate TDA image and text preprocessing																	
2.7 Incorporate TDA into graph (linear layer)																	
2.7 Incorporate TDA into graph (drop nodes)																	
2.8 Run code for results																	
Phase 3: Report redaction																	
3.1 Topology																	
3.2 Stability theorem																	
3.3 Recommenders																	
3.4 LATTICE																	
3.5 Code modification																	
3.6 Results																	
3.7 Conclusions																	
3.8 Introduction & abstract																	

Figure 1.1: **Gantt diagram.** Reflects the temporal planning of the tasks required to complete this project.

- **Chapter 3: Code implementation.** States the model choice and how the code is organized. It details the use of TDA to extract features from product images and descriptions, and their incorporation to the RS. Lastly, it explains how the topological descriptors from the item graph are incorporated into the neural network and used it to prune it.
- **Chapter 4: Results.** Compares seven RSs to decide which one is the best multimodal RS. Moreover, it shows the values obtained when fine-tuning LATTICE and depicts the results obtained by modifying LATTICE.
- **Chapter 5: Conclusions.** Reflects on lessons learned and discusses the results obtained. It also makes some suggestions for future improvements.

Chapter 2

Related work

In this chapter we study the bases of Topological Data Analysis (TDA) (see Section 2.1) and the state of the art of the Recommender Systems (RS) (see Section 2.2); with special attention to LATTICE; that is the one whose implementation we modify to incorporate TDA onto it.

2.1 Topological Data Analysis

TDA is an emerging field that studies data sets from the perspective of Algebraic Topology; and also benefits from combinatorial, analytical and statistical tools. Edelsbrunner et al. (2002) [5] and Zomorodian and Carlsson (2005) [35] popularized persistent homology at the beginning of the decade of 2000, but many consider that the article by Carlsson (2009) [3] consolidated this approach to data analysis.

TDA provides a description of the shape of data and gives qualitative and quantitative information in ways that are robust against noise. In the following points we cover two topological concepts that are basic for TDA: homology and filtrations; and how those are used to compute topological descriptors such as persistence and landscapes. We also prove the Filtering Function Stability Theorem, that explains why topological descriptors are stable under small changes in the data. To cover the bases of Algebraic Topology, we follow [9, Chapter 2].

2.1.1 Homology

Homology was born as a method to define and categorize holes in a manifold, or more generally in a topological space. The rank of the n-th homology group of a space can be interpreted as the number of n-dimensional holes of that space: the

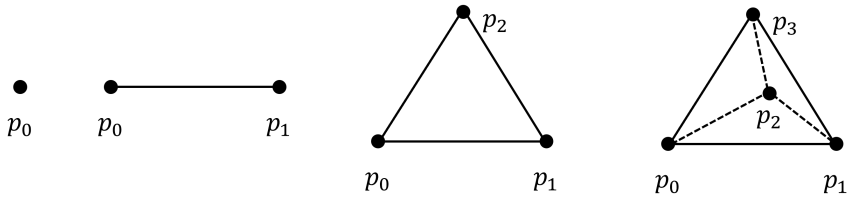


Figure 2.1: **Examples of n-simplices.** From left to right: 0-simplex, 1-simplex, 2-simplex, 3-simplex.

0-th homology group counts the amount of connected components, and the 1-st homology group counts the cardinality of a maximal set of independent cycles.

In this section, we first look at simplicial complexes and how homology is defined on those, and later we study cubical complexes and their homology.

Simplicial complexes

Definition 2.1. An *n-simplex* in $\mathbb{R}^m$, with $0 \leq n \leq m$, is the smallest convex set that contains $n+1$ points $p_0, \ldots, p_n$ that do not lie in a hyperplane of dimension less than n, that is, such that the vectors $\overrightarrow{p_0p_1}, \ldots, \overrightarrow{p_0p_n}$ are linearly independent.

The points p_i are called the *vertices* of the simplex, and the n-simplex is denoted $[p_0, \ldots, p_n]$. The *dimension* of an n-simplex is n. In Fig. 2.1 examples of a 0-simplex, 1-simplex, 2-simplex and 3-simplex are depicted.

Definition 2.2. We refer to an *ordered n-simplex* as an n-simplex with a specified ordering of its vertices. The ordering in $[p_0, \ldots, p_n]$ determines an orientation of the edges $[p_i, p_j]$ according to increasing subindices.

Definition 2.3. We define the *standard n-simplex* Δ^n as the n-simplex whose vertices are the unit points $e_0, \ldots, e_n$ of the coordinate axes, as shown in Fig. 2.2, that is,

$$\Delta^n = [e_0, \ldots, e_n] = \{(t_0, \ldots, t_n) \in \mathbb{R}^{n+1} \text{ such that } \sum_i t_i = 1 \text{ and } t_i \geq 0 \text{ for all } i\}.$$

Definition 2.4. If we delete one of the $n+1$ vertices of an n-simplex $[p_0, \ldots, p_n]$, the remaining n vertices span an $(n-1)$-simplex, called a *face* of $[p_0, \ldots, p_n]$.

Definition 2.5. The union of all the faces of Δ^n is the *boundary* of Δ^n, written $\partial\Delta^n$. The open simplex $\mathring{\Delta}^n := \Delta^n - \partial\Delta^n$ is the *interior* of Δ^n.

Definition 2.6. A *geometric simplicial complex* in $\mathbb{R}^d$ is a set X of simplices in $\mathbb{R}^d$ that is closed under taking faces and intersections.

This means that every face of a simplex in X is in X and any two simplices of X are either disjoint or intersect along one common face.

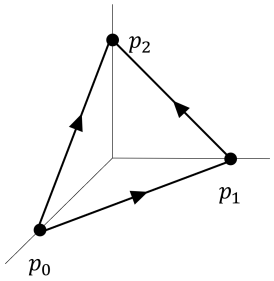


Figure 2.2: **Ordered standard 2-simplex (Δ^2).** Unit points of the coordinate axes are vertices, and the edges have an orientation according to increasing subindices.

Definition 2.7. The *underlying space* of a geometric simplicial complex X is the topological space $|X|$ formed by the union of all the simplices of X with the Euclidean topology.

Definition 2.8. An *abstract simplicial complex* with *vertex set* $V = \{v_i\}_{i\in I}$ is a collection K of nonempty finite subsets $\{v_{i_0}, \ldots, v_{i_k}\} \subseteq V$ such that $\{v\} \in K$ for all $v \in V$ and such that, if $F \in K$ and $G \subseteq F$ is nonempty, then $G \in K$.

The elements of V are called *vertices* of K, and the elements of K are called *faces* of K. For $k \geq 0$, a face $\{v_{i_0}, \ldots, v_{i_k}\}$ is called a *k-face*. Every vertex $v \in V$ is a 0-face and 1-faces are called *edges*. For any m, the collection of all k-faces of K for $0 \leq k \leq m$ is an abstract simplicial complex, called the *m-skeleton* of K.

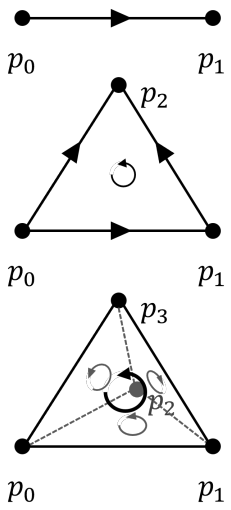
Definition 2.9. An abstract simplicial complex with vertex set $V = \{v_i\}_{i\in I}$ where I is equipped with a total order is called *ordered*.

Definition 2.10. If K is an ordered abstract simplicial complex, with vertex set $V = \{v_i\}_{i\in I}$, the *geometric realization* X_k of K is the geometric simplicial complex defined as follows: let $\{e_0, \ldots, e_n\}$ be the set of coordinate unit points in $\mathbb{R}^{n+1}$. For each k-face $\{v_{i_0}, \ldots, v_{i_k}\}$ in K with $0 \leq k \leq n$, consider the k-simplex $[e_{i_0}, \ldots, e_{i_k}]$ in $\mathbb{R}^{n+1}$, and let X_K be the set of all such simplices associated with the faces of K.

Let K be an ordered abstract simplicial complex with vertex set $V = \{v_i\}_{i\in I}$. We denote its faces $(i_0 \cdots i_k)$ with $i_0 < \cdots < i_k$ instead of $\{v_{i_0}, \ldots, v_{i_k}\}$. For every $n \geq 0$, we write $C_n(K)$ to denote the free abelian group over the set of all n-faces of K.

Definition 2.11. Elements of $C_n(K)$ are called *n-chains* and are written as finite formal sums $\sum_\alpha n_\alpha (i_0^\alpha \cdots i_n^\alpha)$ with coefficients $n_\alpha \in \mathbb{Z}$ and each $(i_0^\alpha \cdots i_n^\alpha)$ is an n-face of K. If the set of n-faces of K is empty, then $C_n(K) = 0$.

We can observe that the boundary of the n-simplex $[p_0, \ldots, p_n]$ consists of the various $(n-1)$-dimensional simplices $[p_0, \ldots, \hat{p}_i, \ldots, p_n]$, where we use $\hat{p}_i$ to de-



$$\partial[p_0, p_1] = [p_1] - [p_0]$$

$$\partial[p_0, p_1, p_2] = [p_1, p_2] - [p_0, p_2] + [p_0, p_1]$$

$$\partial[p_0, p_1, p_2, p_3] = [p_1, p_2, p_3] - [p_0, p_1, p_2] + [p_0, p_1, p_3] - [p_0, p_2, p_3]$$

Figure 2.3: **Boundaries of simplices.** From top to bottom, the boundary of a 1-simplex, 2-simplex and 3-simplex. It uses signs so that all the faces of the simplex are oriented. Signs in the formulas are $+1$ if the orientation of a face coincides with the orientation induced by the simplex on that face, and -1 otherwise.

note the vertex removed from the n-simplex. We insert signs to take orientations into account and orient the faces coherently, as depicted in Fig. 2.3.

Definition 2.12. Given an ordered abstract simplicial complex K, we define a *boundary homomorphism* $\partial_n \colon C_n(K) \to C_{n-1}(K)$ by specifying its values on basis elements as follows:

$$\partial_n[p_0, \ldots, p_n] = \sum_i (-1)^i [p_0, \ldots, \hat{p}_i, \ldots, p_n].$$

Lemma 2.13. *The composition* $C_n(K) \xrightarrow{\partial_n} C_{n-1}(K) \xrightarrow{\partial_{n-1}} C_{n-2}(K)$ *is zero.*

Proof. We have

$$\begin{aligned}\partial_{n-1}\partial_n[p_0, \ldots, p_n] = &\sum_{j<i} (-1)^i (-1)^j [p_0, \ldots, \hat{p}_j, \ldots, \hat{p}_i, \ldots, p_n] \\ &+ \sum_{j>i} (-1)^i (-1)^{j-1} [p_0, \ldots, \hat{p}_j, \ldots, \hat{p}_i, \ldots, p_n] = 0,\end{aligned}$$

because the second sum cancels the first one out if we change i and j. □

Definition 2.14. Algebraically, we have a sequence of homomorphisms of abelian groups

$$\cdots \xrightarrow{\partial_{n+2}} C_{n+1} \xrightarrow{\partial_{n+1}} C_n \xrightarrow{\partial_n} C_{n-1} \xrightarrow{\partial_{n-1}} \cdots \xrightarrow{\partial_2} C_1 \xrightarrow{\partial_1} C_0 \xrightarrow{\partial_0} 0$$

with $\partial_n \partial_{n+1} = 0$ for each n. Such a sequence $\{C_n\}$ is called a *chain complex*. Note that the sequence has been extended by 0 at the end, with $\partial_0 = 0$.

The equation $\partial_n \partial_{n+1} = 0$ is equivalent to the inclusion Im $\partial_{n+1} \subset$ Ker ∂_n.

Definition 2.15. The *n-th homology* of an ordered abstract simplicial complex K chain complex is the quotient group

$$H_n(K) = \text{Ker } \partial_n / \text{Im } \partial_{n+1}$$

from the chain complex $\{C_n(K)\}$. The elements of Ker ∂_n and Im ∂_{n+1} are called *n-cycles* and *n-boundaries* of K, respectively.

The standard algorithm for computing homology of simplicial complexes has a complexity of $O(m^3)$, where m is the total number of simplices in the given simplicial complex [35].

Cubical complexes

Now, we consider structures formed by cubes instead of simplices [28].

Definition 2.16. An *elementary interval* is an interval $[n, n+1]$ (non-degenerate) or $[n, n]$ (degenerate) for $n \in \mathbb{Z}$. In a d-dimensional space, an *elementary cube* C is a product of d elementary intervals. The dimension of the cube is the number of non-degenerate intervals in the product. 0-cubes, 1-cubes, 2-cubes and 3-cubes are called vertices, edges, squares and voxels, respectively.

Definition 2.17. Given two elementary cubes $a, b \subseteq \mathbb{R}^d$, we say that a is a *face* of b if $a \subseteq b$.

Definition 2.18. A *geometric cubical complex* in $\mathbb{R}^d$ is a set of elementary cubes in $\mathbb{R}^d$ that is closed under taking faces and intersections.

Definition 2.19. The *underlying space* of a geometric cubical complex X is the topological space $|X|$ formed by the union of all the cubes of X with the Euclidean topology.

Definition 2.20. A *filtered cubical complex* is a cubical complex X together with a function $f_n \colon X_n \to \mathbb{R}$ for each $n \geq 0$, where X_n denotes the set of n-faces of X.

We represent filtered cubical complexes as shown in Fig. 2.4.

To define homology of cubical complexes, there are two main approaches. The first one is to triangulate each cube, dividing it into simplices and, from that, compute homology as it has already been defined. To triangulate a cube, one can

$f_v(V_0)$	$f_e(E_0)$	$f_v(V_1)$	$f_e(E_1)$	$f_v(V_2)$	$f_e(E_2)$	$f_v(V_3)$
$f_e(E_3)$	$f_c(C_0)$	$f_e(E_4)$	$f_c(C_1)$	$f_e(E_5)$	$f_c(C_3)$	$f_e(E_6)$
$f_v(V_4)$	$f_e(E_7)$	$f_v(V_5)$	$f_e(E_8)$	$f_v(V_6)$	$f_e(E_9)$	$f_v(V_7)$
$f_e(E_{10})$	$f_c(C_3)$	$f_e(E_{11})$	$f_c(C_4)$	$f_e(E_{12})$	$f_c(C_5)$	$f_e(E_{13})$
$f_v(V_8)$	$f_e(E_{14})$	$f_v(V_9)$	$f_e(E_{15})$	$f_v(V_{10})$	$f_e(E_{16})$	$f_v(V_{11})$

Figure 2.4: **Filtered cubical complex.** Each vertex, edge and cube has a value assigned by a function f_v, f_e and f_c respectively.

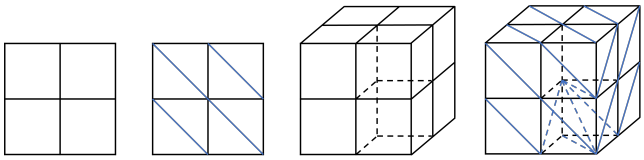


Figure 2.5: **Freudenthal triangulations.** From left to right: 2D cubical complex, its triangulation; 3D cubical complex, its triangulation. This technique can be used to compute homology for cubical complexes, but it is costly, since the amount of simplices to consider is much greater than the number of cubes.

use Freudenthal's triangulations [6], as described in [28] and illustrated in Fig. 2.5. The second approach, which we next describe, aims to avoid the increase in data size that occurs when triangulating, and it is the one used in the `Python` libraries that we use later in the project [27] to compute homology of grayscale images.

Similarly as in the simplicial case, to each cubical complex X one associates a chain complex of *cubical chains*, in which the group of cubical n-chains $C_n(X)$ is defined to be the free abelian group generated by the set X_n of n-faces of X.

A boundary homomorphism from $C_n(X)$ to $C_{n-1}(X)$ is defined as follows [19]. For each elementary cube $\sigma = I_1 \times \cdots \times I_n$ with $I_i = [a_i^0, a_i^1]$ for $i = 1, \ldots, n$ non-degenerate intervals, let

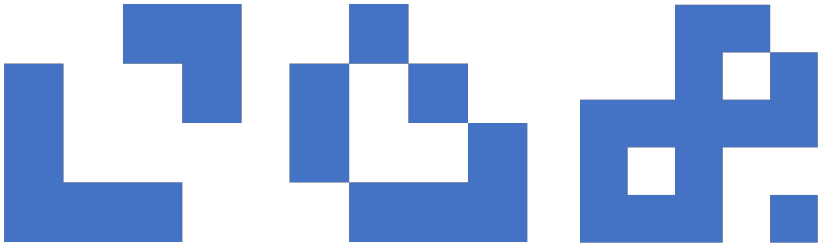
$$\partial_n(\sigma) = \sum_{j=1}^{n} (-1)^j (\lambda_j^0 \sigma - \lambda_j^1 \sigma),$$

where $\lambda_j^i \sigma$ replaces the interval in position j for the degenerate interval $\{a_j^i\} :=$

$[a_j^i, a_j^i]$. In other words, $\lambda_j^i \sigma := I_1 \times \cdots \times \{a_j^i\} \times \cdots \times I_n$.

The above definition of homology corresponds to the *T-construction* discussed in [2]. This is in fact the one implemented in the libraries that we use [27]. Thus, given a 2-dimensional cubical complex, a cycle is formed when a void region becomes encircled in the underlying topological space, as shown in Fig. 2.6.

Indeed, the homology groups of a cubical complex (or of a simplicial complex) are isomorphic to the singular homology groups of the underlying topological space. A detailed proof of this result is given in [9]. An efficient algorithm to compute homology of cubical complexes can be found in [14, Chapter 4].



Cubical complex 1:
2 connected components
0 cycles

Cubical complex 2:
1 connected components
1 cycle

Cubical complex 3:
2 connected components
2 cycles

Figure 2.6: **Cubical complex homology**. Three cubical complexes with their connected components and cycles listed below using a *T*-construction.

2.1.2 Filtrations and filtering functions

In this section we explore filtrations: what they are and how to generate them using filtering functions. Filtrations can be defined for topological spaces, cubical complexes and simplicial complexes, and they involve creating a collection of nested subspaces, with the purpose of examining the unique properties on each subspace and observe their evolution.

Definition 2.21. Given a topological space X and an ordered set I, an I-indexed *filtration* of X is a collection $\{X_i\}_{i \in I}$ of subspaces such that $X_i \subseteq X_j$ if $i \leq j$ and $\bigcup_{i \in I} X_i = X$.

There are similar notions for simplicial complexes and cubical complexes:

Definition 2.22. Given a simplicial complex or a cubical complex K and an ordered set I, an I-indexed *filtration* of K is a collection $\{K_i\}_{i \in I}$ of subcomplexes such that $K_i \subseteq K_j$ if $i \leq j$ and $\bigcup_{i \in I} K_i = K$.

Vietoris-Rips filtration

A *point cloud* is a subset $P = \{p_i\}_{i \in I}$ of a metric space (usually Euclidean).

Definition 2.23. For each value of a parameter $r \in [0, \infty)$, the *Vietoris-Rips complex* of a point cloud P is the simplicial complex defined as

$$VR_r(P) = \{S \subseteq P \text{ such that } S \text{ is finite and } \operatorname{diam}(S) \leq r\},$$

where the *diameter* $\operatorname{diam}(S)$ is the maximum distance between two points in S. Observe that if $r \leq s$ then $VR_r(P) \subseteq VR_s(P)$.

Definition 2.24. The *Vietoris-Rips filtration* is the nested collection of Vietoris-Rips complexes

$$VR(P) = \{VR_r(P)\}_{r \in [0,\infty)}.$$

Definition 2.25. Given a non-negative number r and a graph $G = (V, E)$ with $V = \{v_i\}_{i \in I}$ vertices and $E = \{e_j\}_{j \in J}$ weighted edges, we denote w_{e_j} the weight of the edge e_j. A *thresholded graph* G_r is a graph that only preserves the edges with weight less than r, that is, $G_r = (V, E')$ with $E' = \{e \in E \mid w_e < r\}$.

Given any graph $G = (V, E)$ with undirected weighted edges, we can consider for each $r \in [0, \infty)$ the thresholded graph G_r and, considering the weights in the edges as the distance between nodes, we have successfully adapted the Vietoris-Rips filtration from point clouds to graphs.

As we see in the following Section 2.1.3, to compute persistent homology for a graph we count: (i) how many connected components does $VR_r(G)$ have for $r \in [0, \infty)$; (ii) how many new "holes" (d-dimensional cavities) appear, and (iii) how many of the already existing ones are filled. A d-dimensional cavity is filled when there is a collection of $(d+1)$ simplicies that has the cavity as boundary.

In Fig. 2.7 we show a filtration of a weighted, undirected graph.

For directed graphs, we consider another complex [17] that mimics the process in Vietoris-Rips, but instead uses ordered n-simplices (see Def. 2.2).

Definition 2.26. For a directed graph $G = (V, E)$, the *directed flag complex* $dFl(G)$ is the ordered simplicial complex whose k-simplices are ordered and have dimension $(k+1)$: $\sigma = (v_0, v_1, \ldots, v_k)$ such that $v_i \in V$ for all i and $(v_i, v_j) \in E$ for $i < j$.

We can also consider thresholded graphs G_r for an oriented graph G.

Definition 2.27. The *Flagser filtration* is the nested collection of directed flag complexes

$$F(g) = \{dFl(G_r)\}_{r \in [0,\infty)}$$

where r is a threshold parameter.

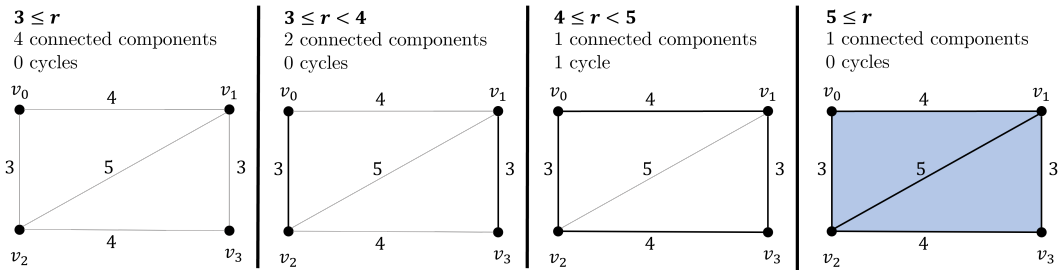


Figure 2.7: **Vietoris-Rips filtration of a weighted undirected graph.** For each value of r, we find, first, 4 connected components that later, for $r = 3$, merge into two. For $r = 4$ we find a cycle that disappears when $r = 5$, because it gets filled with simplices.

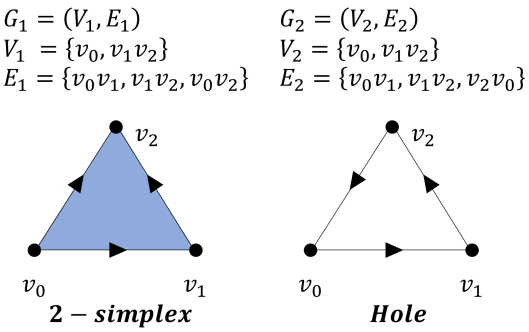


Figure 2.8: **Flagser filtration.** In this picture, two triangular directed graphs are shown. The first one is directed acording to increasing subindices, and this makes it a 2-simplex according to the Flagser filtration. The second graph, instead, is not correctly oriented to be an ordered 2-simplex and this is why it is categorized as a hole, according to the Flagser filtration.

The main difference with non-oriented graphs is that here we consider ordered n-simplices (see Def. 2.2). With that, only if edges are oriented according to increasing subindices we find a simplex; in other cases, the hole remains open. An example is illustrated in Fig. 2.8.

Cubical filtration

With a filtered cubical complex, we can consider the complexes formed by those faces with a value less than r. Similarly to what we had done before, this operation is called thresholding and it allows us to obtain a filtration for the cubical complex. In Section 2.1.3 we count how many connected components and cavities the filtered cubical complex has for each threshold, in order to compute homology.

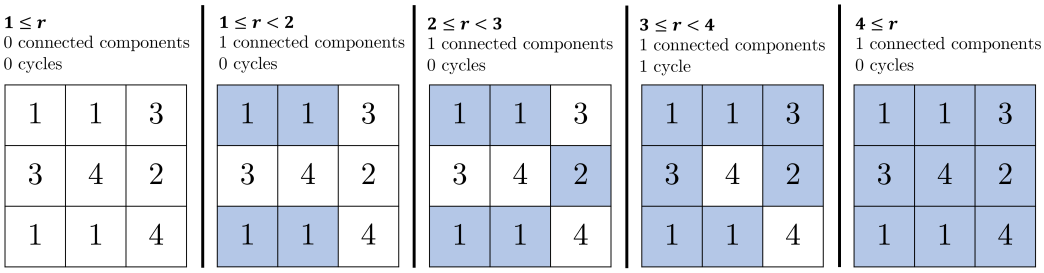


Figure 2.9: **Cubical filtration.** For each value of r we find, first, two connected components, that later merge and at $r = 3$ form a cycle. Lastly, it fills up into a connected component.

Given a grayscale image, we can imagine it as a cubical complex with the values of gray laying in the faces [29]. To fill in the values in edges and vertices, there are many algorithms; one of them being averaging the values of the cubes touching each edge and vertex. Fig. 2.9 depicts the filtration of a cubical complex.

2.1.3 Persistent homology

In this section, we take a look into what is persistent homology and how to represent it.

Definition 2.28. A *filtering function* on a simplicial complex or a cubical complex K is a real-valued function $f\colon K \to \mathbb{R}$ such that $f(a) \leq f(b)$ when $a \subseteq b$. If the range of f is finite, then f determines a finite filtration

$$K_0 \subseteq K_1 \subseteq \ldots \subseteq K_{m-1} \subseteq K_m = K$$

where $\mathrm{Im}(f) = \{\varepsilon_0, \ldots, \varepsilon_m\}$ with $\varepsilon_0 < \ldots < \varepsilon_m$ and

$$K_i = \{a \in K \text{ such that } f(a) \leq \varepsilon_i\} = f^{-1}((-\infty, \varepsilon_i])$$

for each $i \in \{0, \ldots, m\}$. The subcomplexes K_i are called *sublevel complexes* of the filtering function f.

Given a filtration $\{K_i\}$, each inclusion $K_i \hookrightarrow K_j$ of sublevel complexes induces a homomorphism $\varphi_n^{i,j}\colon H_n(K_i) \to H_n(K_j)$ between homology groups (Def. 2.15).

The idea behind persistent homology is to study homological changes of the sublevel complexes of a filtering function. It captures the birth and death times of homology classes across sublevel complexes as it grows from K_0 to K.

Definition 2.29. Given a homology class $\alpha \in H_n(K_j)$, we say that α is *born* at K_j if it does not belong to the image of $\varphi_n^{i,j}$ for any $i < j$. We say that a class $\alpha \in H_n(K_j)$ *dies* at K_j if $\varphi_n^{i,j-1}(\alpha) \neq 0$ and $\varphi_n^{i,j}(\alpha) = 0$.

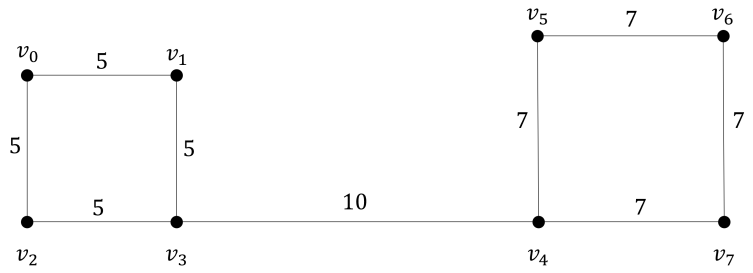


Figure 2.10: **Example graph.** We compute the topological descriptors of this weighted unoriented graph as we define them.

Definition 2.30. The image of $\varphi_n^{i,j}$ is a subgroup of $H_n(K_j)$, called *persistent homology group* and denoted by $H_n^{i,j}(K)$. It contains the homology classes that are born at or before K_i and die at K_j or later. The classes that survive until K are called *essential*.

Definition 2.31. Given a filtration induced by a filtering function and a homology class α born at K_i that dies at K_j, we say that the *life (or persistence)* of α is $\varepsilon_j - \varepsilon_i$.

In the following sections, we see many representations of persistent homology. We accompany them with an example computing persistent homology of the weighted undirected graph shown in Fig. 2.10. This graph has two squares, connected with each other, and we aim to see how the different descriptors capture the evolution of its corresponding homology classes.

Persistence barcodes

From the graph in Fig. 2.10, we can evaluate how many points and cycles there are, depending on the treshold that we consider. We can represent the death and birth of each of them using a *persistence barcode*, that consists of a collection of horizontal line segments in a plane coordinate system whose y-axis has an ordered sequence of homology generators for $H_0, H_1, \ldots$ and whose x-axis has marks at $\varepsilon_0, \ldots, \varepsilon_m$. Given a homology generator α born at K_i that dies at K_j, a horizontal line from ε_i to ε_j is drawn. Essential generators born at K_i are represented as infinite rays $[\varepsilon_i, \infty)$. Each interval indicates the life range of the corresponding homology generator. Fig. 2.11 shows the barcode of the example graph (Fig. 2.10).

It is important to note that the order in which we represent the lines, and which of them is considered *alive* when two of them merge, is chosen arbitrarily. With the following result, we see that it does not affect the end product.

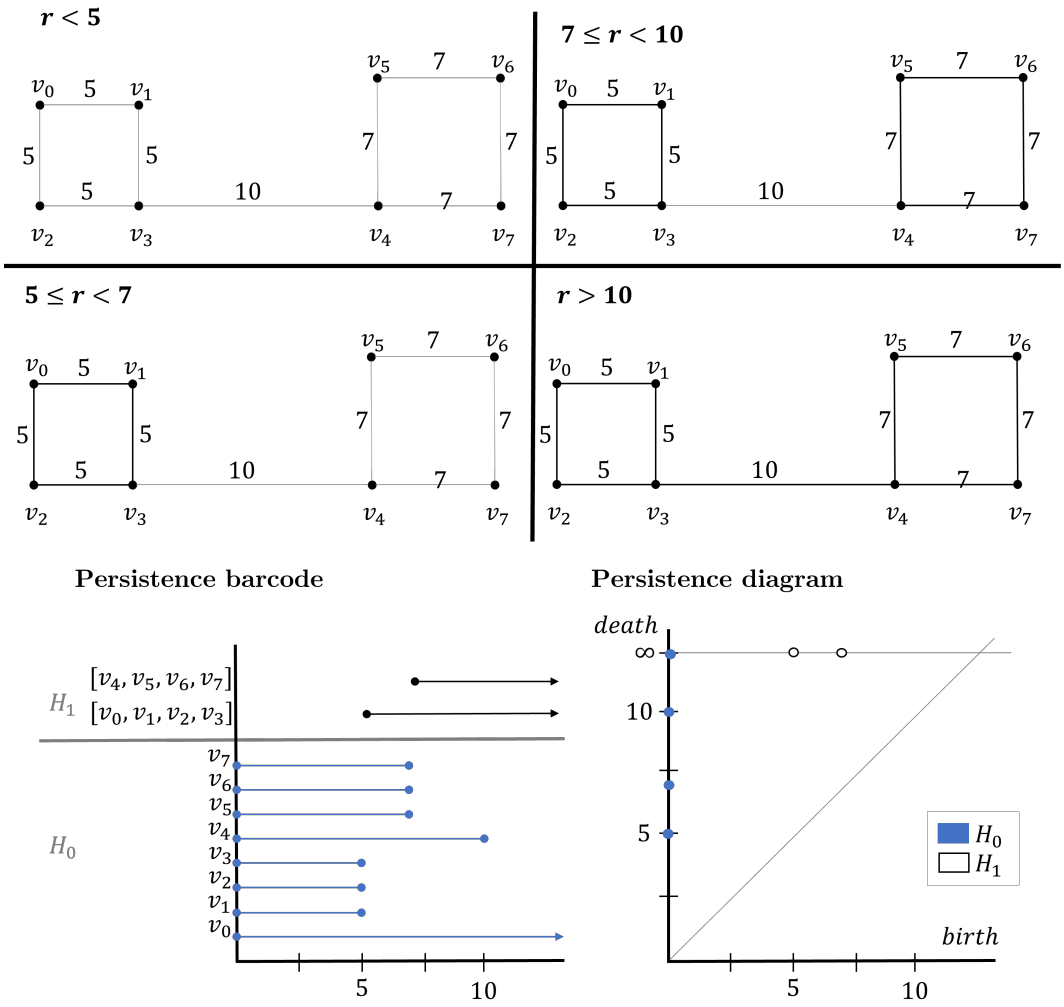


Figure 2.11: **Persistence barcode and persistence diagram of the example graph, using a Vietoris-Rips filtration.** We consider an increasing radius and, for each value, count how many connected components and cycles are in the graph. These values are monitored on a persistence barcode, and we separate them depending on which homology group they belong: H_0 for connected components and H_1 for cycles. On the persistence diagram, each point (b_i, d_i) represents the death and birth of one of the homology generators.

Theorem 2.32 (Normal Form Theorem). *For a field $\mathbb{F}$ and an interval $I = [a, b)$ we define a persistence module (see Def. 2.42)*

$$\mathbb{F}[a,b)_t = \begin{cases} \mathbb{F} & \text{if } t \in [a,b) \\ 0 & \text{otherwise,} \end{cases} \qquad \pi_{s,t} = \begin{cases} \mathbb{1} & \text{if } s,t \in [a,b) \\ 0 & \text{otherwise.} \end{cases}$$

If (V, φ) is a persistence module of finite type, then there exists a finite collection of intervals $\{(I_i, m_i)\}_{i=1,\dots,N}$ of intervals $I_i = [a_i, b_i)$ or $I_i = [a_i, \infty)$ where $I_i \neq I_j$ for $i \neq j$, with multiplicities m_i, such that

$$V \cong \bigoplus_{i=1}^{N} \mathbb{F}(I_i)^{m_i}$$

as persistence modules. Moreover, this data is unique up to permutations, that is, to any persistence module of finite type there corresponds a unique barcode with intervals I_i and multiplicities m_i.

Proof. See [20, Theorem 2.1.1]. □

We can encode the length of the bars as a *multiset*: a set formed by pairs (value, cardinality). A multiset can also be thought as a modification on the definition of "set" that allows repetiton of elements. From this theorem, we can deduce that the number of bars of a barcode and the unordered multiset of their lengths does not depend on the choice of homology generators along the given filtration.

Persistence diagrams

We can now transform each barcode into a *persistence diagram*, which has a point (b_i, d_i) in a coordinate plane for each segment in the barcode starting at b_i and ending at d_i. Infinite rays $[b_i, \infty)$ are represented as points (b_i, ∞), where ∞ is a fixed value larger or equal than the largest value ε_m of the filtering function.

This means that any point (b_i, d_i) in a persistence diagram represents the life range of a homology generator from the filtered simplicial complex that originated the barcode. In our example (Fig. 2.10), all the H_0 generators are born at 0 and the ones that correspond to the leftmost square die at $t = 5$, the ones in the other square die at $t = 7$ and lastly, at $t = 10$ both components merge and only two cycles are left. This is why all the dimension 0 points in the persistence diagram are in the vertical axis and all the dimension 2 are on the infinity line. We can see the persistence diagram corresponding to the example graph in Fig. 2.11.

We can extract *topological descriptors* from the persistence representations that we have just defined. There are two kinds of descriptors: numerical and vector-valued. The first type includes those that are numerical values, while the second

one may contain discretized continuous functions. Persistence barcodes and diagrams are hard to translate into computer programs, since their representation does not match conventional data structures. Numerical and vector-shaped descriptors, on the other hand, are easier to store, and this makes them more suitable when analyzing data. We use all of the descriptors that we define in the following lines later in this project (see Chapter 3) to extract information from images, text and item graphs.

Numerical topological descriptors

Definition 2.33. *Total persistence* of a persistence diagram $\{(b_i, d_i)\}_{i\in I}$ is defined as

$$P = \sum_i (d_i - b_i).$$

It is a numerical descriptor that responds to the accumulated life time of all the given points, counting multiplicities.

In the case of the example graph (Fig. 2.10), we can compute total persistence for our homology classes in H_0: $P_0 = (5-0) + (5-0) + (5-0) + (7-0) + (7-0) + (7-0) + (10-0) = 46$. For the homology classes in the homology group H_1, $P_1 = \infty$, since all points have infinite persistence.

Definition 2.34. The *mean lifetime* is computed as the average persistence from a persistence diagram $\{(b_i, d_i)\}_{i\in I}$, that is,

$$ML = \frac{\sum_i (d_i - b_i)}{\#I}.$$

Considering the life of all the homology classes that have dimension 0, their mean lifetime is $ML_0 = P/\#I = 46/7 = 6.57$.

Definition 2.35. The *standard deviation* of the persistence is

$$STD = \sqrt{ML} = \sqrt{\frac{\sum_i (d_i - b_i)}{\#I}}.$$

This numerical descriptor takes value $STD_0 = 0.84$ for the classes that have dimension 0.

Definition 2.36. The *entropy* from a persistence diagram $\{(b_i, d_i)\}_{i\in I}$ is defined as

$$E = \sum_i \left(\frac{d_i - b_i}{P}\right) \log_2 \left(\frac{d_i - b_i}{P}\right).$$

As we see in Proposition 2.37, entropy indicates how different the bars in the associated barcode are in length. It ranges from $[0, \log_2 n]$, and only reaches the upper bound when all the points coincide, while it gets lower when the lifetime of the homology classes differs more from the average lifetime.

In our example, we can compute this topological descriptor for the homology classes in H_0 and obtain $E_0 = 1.91$.

Proposition 2.37. *Given a persistence diagram with n points, the entropy associated to it takes values in the range* $[0, \log_2 n]$. *It reaches* $\log_2 n$ *when all the points in the diagram are the same.*

Proof. We consider the inequality

$$\log_2\left(\frac{\sum x_j}{n}\right) \leq \frac{\sum x_i \log_2 x_i}{\sum x_j} \tag{2.1}$$

for $x_1, \ldots, x_n$ positive real numbers. It implies that $E \leq \log_2 n$ in all cases, and it is only equal when all x_i are equal. To see that the equation 2.1 is true, we can multiply both sides by $\sum x_j$ (which is positive since all x_i are positive) and obtain

$$\sum x_i \log_2\left(\frac{\sum x_j}{n}\right) \leq \sum x_i \log_2 x_i.$$

This inequality holds since $n \geq 1$. □

Vector-valued topological descriptors

The following descriptors capture persistence through vectorial representation, depicting homology group evolution with continuous functions.

Definition 2.38. Given a persistence diagram $\{(b_i, d_i)\}_{i \in I}$, the *Betti curve* is the graph of the function $f\colon \mathbb{R} \to \mathbb{N}$ defined as $f(t) = \#\{i \in I \mid b_i \leq t \leq d_i\}$.

A Betti curve counts how many homology generators are *alive* at each given moment in time. The Betti curve for our example graph is depicted in Fig. 2.12.

Definition 2.39. Given a pair (b, d), we define a corresponding *tent function* as $\Lambda_{(b,d)}(t) = \max\{0, \min\{\, t - b, d - t\}\}$. It is shaped as Fig. 2.13 shows.

Definition 2.40. Given a positive integer k, we define the *k-th landscape* from a persistence diagram $\{(b_i, d_i)\}_{i \in I}$ as a function $\lambda_k\colon \mathbb{R} \to \mathbb{R}$ defined as $\lambda_k(t) = kmax\{\Lambda_{(b_i,d_i)}(t)\}$, where $kmax$ returns the k-th largest value of a multiset, or zero if k is larger than the size of the multiset.

To easily compute the landscapes of our example, we rotate the persistence diagram (Fig. 2.11) and lay it against the diagonal. To plot the tent function accurately, it is only required to re-scale the axis so that the birth and death of each generator are correctly placed. For the first landscape of the homology classes in

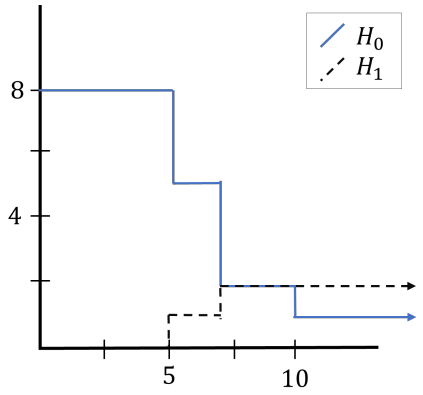


Figure 2.12: **Betti curve of the example graph.** It shows how many homology generators are alive at each moment. We have computed a Betti curve for generators in H_0 (straight line) and another one for generators in H_1 (dotted line).

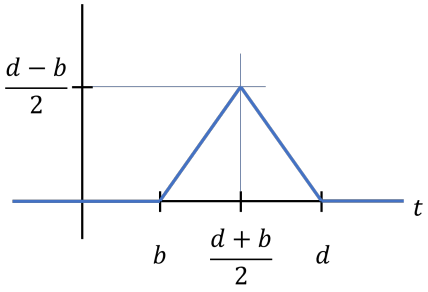


Figure 2.13: **Tent function.** $\Lambda_{(b,d)}(t) = \max\{0, \min\{\ t-b, d-t\}\}$.

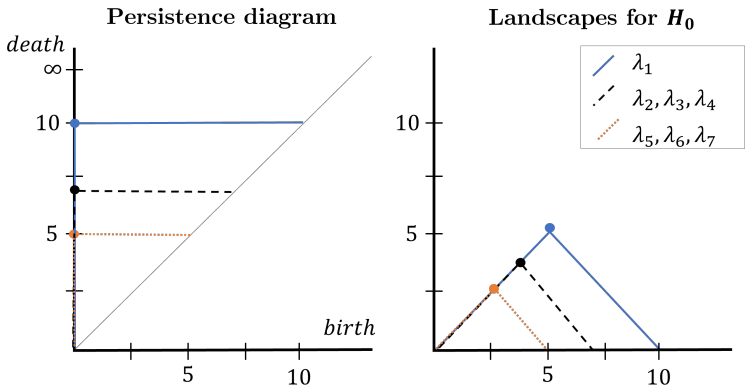


Figure 2.14: **Landscapes of the example graph.** On the left, the persistence diagram for the generators in H_0. On the right, the landscapes for the first seven generators. We see that, since the multiset is $\{(10,1),(7,3),(5,3)\}$, the *kmax* function returns 0 for every $k > 7$. Therefore, all the k-landscapes with $k > 7$ are the constant function $\lambda_k = 0$.

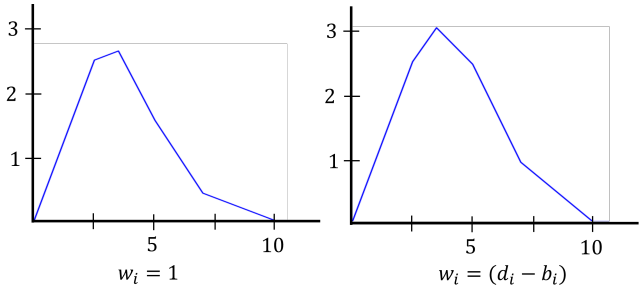


Figure 2.15: **Silhouettes of the example graph:** Silhouette with constant weight (left) and quadratic weight (right).

H_0 we take the tallest tent, and for the second one, the second tallest. This process is portrayed in Fig. 2.14.

We transform each of the landscapes into numerical descriptors by computing the area under them. We use this in our code, as translating functions into vectors involves discretizing them, which is inconvenient.

Definition 2.41. A *silhouette* from a persistence diagram $\{(b_i, d_i)\}_{i\in I}$ is a weighted average of corresponding tent functions with weights $w_i > 0$ for all $i \in I$, that is,

$$\varphi(t) = \frac{1}{\sum w_i} \sum w_i \Lambda_{(b_i,d_i)}(t).$$

A frequent choice is $w_i = (d_i - b_i)^p$ with $p > 0$. Choosing a smaller p enhances low-persistence features. The silhouette for the example function, with weight constant $w_i = 1$ and quadratic: $w_i = (d_i - b_i)^2$ is shown in Fig. 2.15. We can transform them into numerical descriptors by considering the area under each silhouette.

We can follow the same steps with the cubical complex portrayed in Fig. 2.16. Once we obtain the barcode representation, the steps that follow are identical to the ones that we have just gone through for the directed graph. The total persistence for both homology groups is 1, and since there is only one point in each of them, the mean lifetime coincides with the persistence. The entropy has its maximum possible value: $\log_2(1)$, since all points coincide (because there is only one point on each case).

2.1.4 Filtering function stability

In this section, we study the stability theorem for functions, that explains why topological descriptors are robust against noise. The stability theorem states that

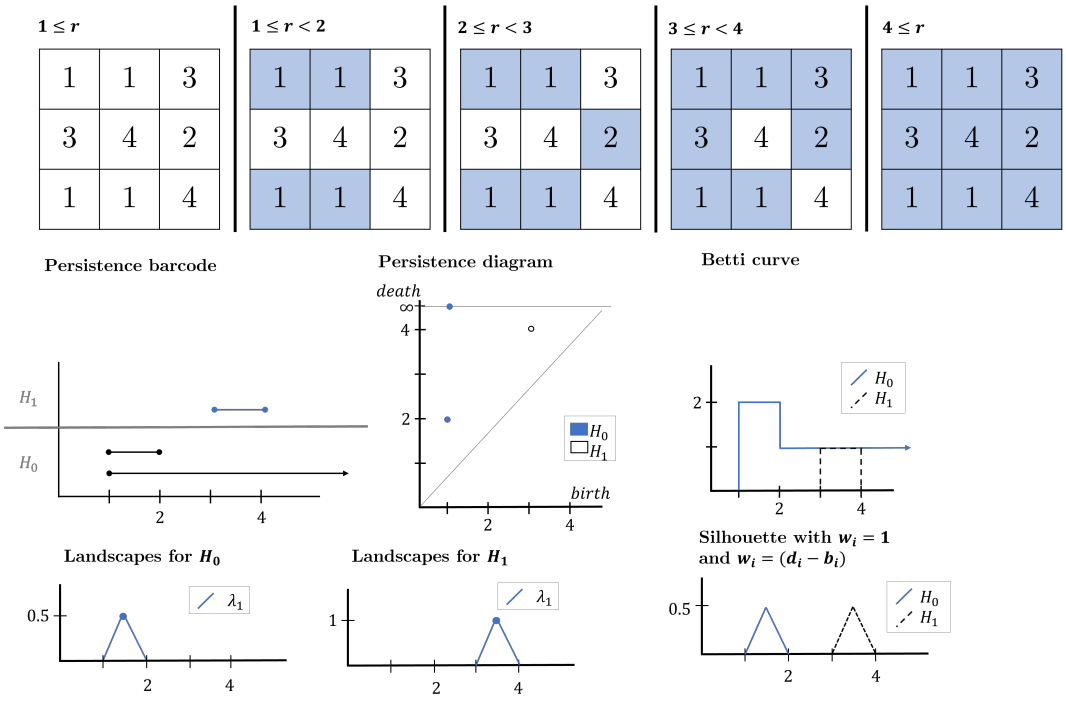


Figure 2.16: **Cubical filtration descriptors.** On this image we portray the cubical filtration for a weighted cubical complex, and compute its barcode and persistence diagram. From that, we extract the corresponding Betti curve and the first landscape for both H_0 and H_1. It is important to realise that, since there is only one homology generator for each category that has not infinite persistence, all landscapes λ_k with $k > 1$ are the constant function $\lambda_k = 0$. Lastly, since both points have persistence 1, their silhouettes coincide for $w_i = 1$ and $w_i = (d_i - b_i)^2$.

the difference between two persistence modules is bounded by the distance between the filtering functions that generate them. To prove this theorem, we use the following result.

Isometry Theorem

Definition 2.42. A *persistence module* over $\mathbb{R}$ is a pair (V, π) where $V = \{V_t\}_{t\in\mathbb{R}}$ is a collection of $\mathbb{R}$-vector spaces and π is a collection of $\mathbb{R}$-linear maps $\pi_{s,t} \colon V_s \to V_t$ for $s \leq t$, called *translations*, such that $\pi_{s,t} \circ \pi_{r,s} = \pi_{r,t}$ if $r \leq s \leq t$ and $\pi_{t,t} = id$ for all $t \in \mathbb{R}$. A persistence module (V, π) is of *finite type* if V_t is finite-dimensional for all t and the sequence $\{V_t\}$ is constant except for a finite number of t values.

Definition 2.43. Two persistence modules (V, π) and (V', π') are *δ-interleaved* for

$\delta \geq 0$ if there exist morphisms $f\colon V \to V'[\delta]$ and $g\colon V' \to V[\delta]$ such that

$$g[\delta] \circ f = \sigma_{2\delta} \text{ and } f[\delta] \circ g = \sigma'_{2\delta},$$

where we denote $V[\delta]_t = V_{t+\delta}$, $(\sigma_\delta)_t = \pi_{t,t+\delta}$ and $f[\delta]_t = f_{t+\delta}$ for all $t \in \mathbb{R}$.

Definition 2.44. Given two persistence modules (V, π) and (V', π') of finite type with $V_\infty \cong V'_\infty$, we define the *interleaving distance* between them as

$$d_{\text{int}}(V, V') = \inf\{\delta \geq 0 \mid (V, \pi) \text{ and } (V', \pi') \text{ are } \delta\text{-interleaved}\}.$$

Definition 2.45. Given two persistence diagrams D and D' that have the same number of points with infinite death coordinate, a *matching* between D and D' is a bijective function $\varphi\colon D \to D'$, where diagonal points are counted with infinite multiplicity. We define

$$||\varphi|| = \max_{(x,y)\in D} \{d_\infty((x,y), \varphi(x,y)).$$

If the second coordinate is infinite, we adopt $d_\infty((x,\infty),(x',\infty)) = \mid x - x' \mid$.

The *bottleneck distance* between D and D′ is defined as

$$W_\infty(D, D') = \min_{\varphi\colon D \to D'} \{||\varphi||\}.$$

Theorem 2.46 (Isometry Theorem). *Given two persistence modules (V, π) and (V', π') of finite type with $V_\infty \cong V'_\infty$, the interleaving distance between them is equal to the bottleneck distance between their respective persistence diagrams D and D', that is,*

$$d_{\text{int}}(V, V') = W_\infty(D, D').$$

Proof. We refer to [4, Theorem 4.4]. □

Sublevel sets of real-valued functions

Definition 2.47. A *sublevel set* of a function $f\colon X \to \mathbb{R}$, where X is a topologial space, is defined as

$$L_t(f) = \{x \in X \mid f(x) \leq t\} = f^{-1}((-\infty, t]).$$

Note that if f is bounded, then $L_t(f) = \emptyset$ if $t < \inf(f)$ and $L_t(f) = X$ if $t > \sup(f)$. It is also important to note the inclusion $L_s(f) \subseteq L_t(f)$ if $s \leq t$.

Given a topological space X, each function $f\colon X \to \mathbb{R}$ yields a persistence module (see Def. 2.42) $V(f)$ defined as

$$V_t(f) = H_*(L_t(f)) \coloneqq \bigoplus_{n=0}^{\infty} H_n(L_t(f)),$$

where homology is meant with coefficients in $\mathbb{R}$. The maps $\pi_{s,t}\colon V_s(f) \to V_t(f)$ are induced by the inclusions $L_s(f) \hookrightarrow L_t(f)$ if $s \leq t$.

The persistence module $V(f)$ is of finite type if $f\colon \mathbb{R}^d \to \mathbb{R}$ is defined as $f(x) = d(x, P)$ where P is a point cloud in $\mathbb{R}^d$, and also if $f\colon X \to \mathbb{R}$ where X is the underlying space of a simplicial complex or a cubical complex, amongst other cases. The latter case is particularly interesting for this project, since it is the case that we have studied and, with the following theorem, we have a guarantee on the robustness of the descriptors that we have computed. Note that a filtering function $K \to \mathbb{R}$ on a simplicial complex or a cubical complex K yields a function $f\colon |X_K| \to \mathbb{R}$ on the corresponding underlying space, and the sublevel sets of f are underlying spaces of the sublevel complexes of the given filtering function.

Now we can finally state and prove the stability theorem for functions.

Theorem 2.48 (Stability Theorem). *Given a topological space X and two functions $f, g\colon X \to \mathbb{R}$ whose persistence modules $V(f)$ and $V(g)$ are of finite type, we have*

$$W_\infty(D(V(f)), D(V(g))) \leq ||f - g||_\infty.$$

Following with the previous notation, $D(V(f))$ and $D(V(g))$ are persistence diagrams of $V(f)$ and $V(g)$ respectively.

Proof. We prove that

$$d_{\text{int}}(V(f), V(g)) \leq ||f - g||_\infty,$$

which is sufficient, in view of the Isometry Theorem 2.46.

We can consider $\delta = ||f - g||_\infty$ and prove that $V(f)$ and $V(g)$ are δ-interleaved (see Def. 2.44) and we obtain that

$$d_{\text{int}}(V(f), V(g)) \leq \delta$$

as a direct consequence, since d_{int} is defined as an infimum. Since f and g are bounded (because $L_t(f) = L_t(g) = X$ when $t > \max\{\sup(f), \sup(g)\}$), we have that $\dim V_\infty(f) = \dim V_\infty(g)$. One can also observe that

$$V(f)[\delta] = V(f - \delta)$$

since $V[\delta]_t = V_{t+\delta} = H_0(L_{t+\delta}) = H_0(L_t(f-\delta)) = V_t(f-\delta)$ because $L_t(f-\delta) = \{x \in [a,b] \mid f(x) - \delta \leq t\} = \{x \in [a,b] \mid f(x) \leq t+\delta\} = L_{t+\delta}(f)$. Similarly,

$$V(g)[\delta] = V(g-\delta).$$

Since we have defined $\delta := ||f-g||_\infty$, it is immediate that $||f(x) - g(x)|| \leq \delta$ for all $x \in [a,b]$. We can re-write it as

$$g(x) - \delta \leq f(x) \leq g(x) + \delta, \qquad f(x) - \delta \leq g(x) \leq f(x) + \delta,$$

for all x. Therefore, if $f(x) \leq t$, then $g(x) \leq \delta + t$, and we obtain the following inclusions:

$$L_t(f) \hookrightarrow L_{t+\delta}(g) \text{ and } L_t(g) \hookrightarrow L_{t+\delta}(f)$$

for all $t \in \mathbb{R}$, which induce morphisms of persistence modules

$$F\colon V(f) \to V(g)[\delta] \text{ and } G\colon V(g) \to V(f)[\delta].$$

Lastly, to see that they are δ-interleaved we want to see that $G[\delta] \circ F = \sigma_{2\delta}(f)$ and that $F[\delta] \circ G = \sigma_{2\delta}(g)$. The first one is obvious because $\sigma_{2\delta}(f)$ is induced in homology by the inclusion $L_t(f) \hookrightarrow L_{t+\delta}(g) \hookrightarrow L_{t+2\delta}(f)$; and for the second one we can reproduce the same reasoning. Thus the proof is complete, since $V(f)$ and $V(g)$ are δ-interleaved by means of F and G. □

2.2 Recommender Systems

Recommender Systems support users by suggesting items that might be of interest to them [24]. This is typically done by leveraging behavioural patterns from historical data, often in the form of user-item interactions. The information about previous interactions with products may come in many forms: 1-5 star rankings, boolean values that describe if the user has bought it or not, has clicked on it or not, etc. Recommender Systems are usually categorized depending on how they analyze the data sources to create well-suited user-item pairs: *collaborative filtering systems* analyze historical interactions, and *content-based filtering systems* combine those interactions with profile-based information. In this project we focus in the first kind.

Collaborative filtering systems have evolved throughout the years: on first formulations only correlation statistics and predictive modelling were used, but it has progressed and it incorporates methods and algorithms from Machine Learning (ML) and Data Mining (DM), amongst other fields. The research field of RS is continuously changing since it has a lot of applications in the entertainment and

marketing industry: improving the suggestions that the users are presented leads to more engaging on their end.

Formally, the CF is defined as: given a user u from a set of users U and a set of items or products I; we want to rank those products according to how much will the user like them. In order to do that, we frequently use previous interactions of the user u with products, interactions of other users with those products and how similar is u to the other users; in order to make a prediction on the likeliness of the user u to each product in I [18].

We pay special attention to multimodal systems [25]; those are included into the collaborative filtering category and take into account more than one modality in which the users interact with items; such as linguistically (writing a review or comment), visually (images or video), spatially (where they interact), gesturally (body movement, facial expressions) or aurally (music, noises). These are best suited for situations with a great variety of input amongst many clients, tasks, situations and context; they tend to have higher accuracy since they can look into different modalities and give consistency to the user feedback.

We also look into content-aware Recommender Systems, that take into account additional information such as textual descriptions of the items, images and metadata. This helps improve the model, since it allows to better understand the products recommended.

2.2.1 Foundations of Recommender Systems and representation

In this section we establish some common concepts that are found in Recommender Systems; in order to make the detailed exploration of RSs in the next section, more easily understandable.

Definition 2.49. Let $U = \{u_i\}_{i \in I}$ be a set of users, and $P = \{P_j\}_{j \in J}$ a set of products. We define the *user-item interaction matrix* as $R \in \mathbb{R}^{\#I \times \#J}$; where $R_{i,j}$ represents the interaction of the user i and the item j; commonly corresponding to the rating the user has given to it. If one user has not interacted with an item, the interaction is usually given a value that cannot be given as a rating; usually being 0 or -1 or denoted as $\perp$.

Definition 2.50. A *neural network (NN)* is a Machine Learning technique that mimics the human brain. It is constructed as a set of neuron layers, and the output of the n-th neural layer is the input for the n+1-th layer. Each layer is formed by an amount of nodes or neurons, and each of them contains one or more *trainable parameters* that, given an input, weigh it and generate an output. These parameters are trained to minimize the error: given an input and a known result, the input is fed into the network and the output obtained is compared with the result of the

operation we expect the neural network to perform. Then, the trainable parameters are modified slightly and the process is repeated over, until the difference is small enough. Each repetition is called an epoch. Finally, after training, the NN is used to make predictions on new values.

To train a neural network, it is very common to take pairs formed by an input and a known result. Moreover, the training instances are separated into three sets: training, validation and test. The first two of them are used during the training process, and the last one during evaluation. The workflow is the following: on each epoch, we take all the (input, result) pairs in the training set and feed them to our network, this allows it to learn how to classify them, but we have to avoid *overfitting* (performing very well on the values in the training set, but poorly on any other input). To prevent overfitting, after every epoch, we use the validation set and modify the network accordingly. Lastly, after the training phase, we evaluate the model using the test set: depending on how properly it classifies the instances. The evaluation may be done with different metrics, such as precision or recall, see Section 4.2.

Definition 2.51. An *activation function* is the function used to compute the output of a neuron in a neural network. Once the trainable weights in the neuron have been used to modulate each of the inputs, the activation function modifies the value before sending it to the next layer. Examples of that are:

- **Indentity:** $f(x) = x$.
- **Binary Step** $f(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ 0 & \text{if } x < 0. \end{cases}$
- **Rectified Linear Unit (ReLU):** $f(x) = \begin{cases} x & \text{if } x \geq 0, \\ 0 & \text{if } x < 0 \end{cases} = \max\{0, x\}$.
- **Leacky Rectified Linear Unit (Leacky ReLU):** $f(x) = \begin{cases} x & \text{if } x \geq 0, \\ 0.01x & \text{if } x < 0. \end{cases}$

Definition 2.52. A *Convolutional Neural Network (CNN)* is a modification to a NN that addresses the issue of exploding or vanishing gradients. This change involves optimizing filters or kernels for convolutional operations, helping overcome gradient-related challenges. The hidden layers of a CNN include convolution layers, which perform a dot product on the tensor from the previous layer with a matrix. This reshapes the data and abstracts features and local patterns, making CNNs handy for handling structured grid data.

Definition 2.53. A *Graph Convolution Network (GCN)* is similar to a CNN but instead of grid values, it operates on graphs, where the nodes represent values and the edges, relationships between them. The convolution operation is adapted by combining the information from the neighbouring nodes into each of them.

Definition 2.54. Given a graph $G = (V, E)$, we define its *adjacency matrix* (A) as the matrix that represents the graph. If the edges are not weighted, A_{ij} is 1 if nodes i, j are connected, 0 if they are not. If the graph is weighted, A_{ij} contains the value of the edge if they are connected and, if they are not linked, it contains a value that is not valid, such as 0, -1 or ∞. It is important to note that non-directed graphs have a symmetrical adjacency matrix.

Definition 2.55. Given a non directed graph $G = (V, E)$, we define its *Laplacian matrix* as

$$L = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

with A being the adjacency matrix associated with the graph, and D being the diagonal degree matrix, such that D_{ii} is the degree of the node i.

2.2.2 Recommendation models

In this section we look into seven different recommendation models. Note that different recommendation models have been proposed in the literature. Here, we focus on those that have been analysed in the work. The first three of them are of the collaborative filtering kind (MF, NGCF, LightGCN), the following three of them are deep content-aware models (VBPR, MMGCN, GRCN). Finally, we deeply look into LATTICE; that combines collaborative filtering with modality-aware graph structures and convolutions. In fact, this is the model chosen to include TDA.

Matrix Factorization (MF)

It decomposes the user-item interaction matrix R as the product of two lower-dimensional matrices, representing users and items [15]. It uses the matrix that results of the product to predict the ratings that are missing in the first one. To do so, it proceeds as follows:

Let $R \in \mathbb{R}^{\#I \times \#J} := \mathbb{R}^{n \times m}$ to be the user-item interaction matrix, and let $k << n, m$ be a fixed, positive integer, called the latent factor size. Let $P \in \mathbb{R}^{n \times k}, Q \in \mathbb{R}^{k \times m}$. We denote it $\hat{R}$ as the predicted matrix. We observe that the product of matrices $\hat{R} := P \times Q$ generates a matrix that is the same size as the rating matrix, and proceed to minimize the difference between the known values in R and the

computed as a result of the product.

$$\arg\min_{P,Q} \sum_{i,j} ||R_{i,j} - \hat{R}_{i,j}||$$

where $\arg\min_x f(x)$ represents the values of x for which the function $f(x)$ attains a minimum. Then, we can predict the empty values in the original matrix with the values computed as the product of P and Q. To make it more precise, it can incorporate a bias vector and the minimization can be modulated by a regularization term to prevent overfitting.

Neural Graph Collaborative Filtering (NGCF)

Incorporates graph neural networks into collaborative filtering by modeling user-item interactions into a graph and uses it to guess unknown interactions [30].

It is constructed as a three step process:

1. **An embedding layer:** initializes user and item embeddings (we can think of it as a d-dimensional vector that represents features of each item and user).

2. **Multiple embedding propagation layers:** refines the embedding and injects high-order connectivity-relations; by doing the following:

   (a) First-order propagation: given a user-item pair (u, i), it defines the message from i to u

   $$m_{i\to u} = f(e_i, e_u, p_{ui}) = \frac{1}{\sqrt{|N_u||N_i}}(w_1 e_i + w_2(e_i \odot e_u)$$

   where e_i is the item embedding, e_u is the user embedding and p_{ui} is a coefficient used to control the decay. It encodes the interaction between e_i and e_u via the element-wise product and it controls the decay with $p_{ui} = \frac{1}{\sqrt{|N_u||N_i}}$, where N_u and N_i are the direct neighbours of e_u and e_i, respectively. Then, it aggregates the messages into the user representation using a LeackyReLU (see Def. 2.51) as the activation function. We denote the user representation for layer l as $e_u^{(l)}$. In this first-order propagation we obtain

   $$e_u^{(1)} = LeackyReLU(m_{u\to u} + \sum_{i\in N_u} m_{i\to u}).$$

   (b) High-order propagation: stacks l propagation layers and saves the output user and item representation for each of them; so that each of them

have information from their neighbours that are at a distance l or less. If we define the message as

$$\begin{cases} m_{i\to u}^{(l)} & = p_{ui}(W_1^{(l)} e_i^{(l-1)} + W_2^l (e_i^{(l-1)} \odot e_u^{(l-1)})) \\ m_{u\to u}^{(l)} & = W_1^l e_u^{(l-1)} \end{cases}$$

with $W_1^{(l)}, W_2^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ trainable transformation matrices. Then we have the following representation for user u in the $l - th$ step

$$e_u^{(l)} = LeackyReLU(m_{u\to u}^{(l)} + \sum_{i \in N_u} m_{i\to u}^{(l)}).$$

3. **A prediction layer:** aggregates the refined embeddings and outputs an affinity score for a user-item pair. It does so by concatenating the item and user representations obtained by each of the L layers and estimates the users' preference towards the target item as the product of the user and item vectors.

LightGCN

It is a simplified version of a collaborative filtering that uses a low-complexity model for more efficient training [12]. It focuses on aggregating embeddings from user-item interactions in a graph.

To do so, it makes the convolution of the graph iteratively; meaning it aggregates the features of neighbors as the new representation of a target node; without the trainable values that the function in previous models used. The first step is to choose an aggregation function, and then compute each embedding:

$$e_u^{(k+1)} = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u||N_i|}} e_i^{(k)},$$

$$e_i^{(k+1)} = \sum_{i \in N_i} \frac{1}{\sqrt{|N_u||N_i|}} e_u^{(k)}.$$

Since the only trainable parameters are the embeddings at the 0-th layer, this design is very optimal time-wise.

As the last step, all the embeddings are combined as a weighted sum and the prediction is defined, once again, as the product between each user and item embedding.

Visual Bayesian Personalized Ranking (VBPR)

VBPR incorporates visual features into the recommendation process [11]. It considers both user preferences and visual content features of items to provide more personalized recommendations.

It is built on top of a Matrix Factorization model (see Section 2.2.2); and the main difference is that convolves the original product embedding with visual features, which are extracted by a pre-trained CNN from images associated with each of the products. This process also helps reduce dimensionality.

Multi-Modal Graph Convolutional Networks (MMGCN)

MMGCN integrates graph convolutional networks with multi-modal information [13]. It can handle various types of data, such as text, images, and other modalities, to make recommendations.

Before entering the information into the GCN, it encodes contextual information into the raw feature representation of each item, using different approaches depending on which modality it is retrieving the information from. It can also embed more information, such as the source of the feedback, etc.

Once the representation vectors are ready, with all the information embedded, the graph construction begins. It considers the embedding vectors as nodes in the space, and it builds edges depending on modality similarity amongst other characteristics. To weight these edges, it uses angular similarity between nodes. If the nodes come from different modalities, this similarity value is weighted by a trainable parameter. Then, the graph convolution starts; it uses two hyperparameters and a matrix of trainable parameters in order to weight the laplacian matrix and the values coming from the previous layer.

Lastly, it concatenates the initial representations with the features encoded by the GCN, to obtain the final representation.

Graph-Regularized Convolutional Networks (GRCN)

GRCN leverages graph regularization to incorporate content information into the recommendation model [31]. It aims to capture the relationships between items in a graph structure. To do so, it follows a three-step process:

1. **Graph refining layer:** looks to delete noisy edges and only keep the ones that truly represent user preferences. It uses a LeackyRELU as the activation function, and many trainable parameters to adapt the item representation according to the liking of a fixed user. Then, for the same user, it computes the

similarity between each product and the fixed user, to later create a new representation for the user that takes those similarities into account. It iterates this process, updating the user representation on each step. To finish this step, it softly prunes noisy edges, by incorporating the similarity between the user and the item.

2. **Graph convolutional layer:** it stacks L layers in order to propagate high-order information between users and items, like many other models that we have seen before.

3. **Prediction layer:** lastly, it concatenates the output of the previous layer for each modality, both for users and items, and predicts the affinity as the product of those two vectors.

LATent sTructure mining method for multImodal reCommEndation (LATTICE)

LATTICE focuses on modelling item relationships, and it does so by finding high-order item-user-item co-occurrences. It takes both U and I item sets, and an interaction matrix with the feedback from users. It also uses multimodal information from items, coming from text and image, and represented as a vector of d_v features for images and d_t features for text. All these features are used in its five steps, as depicted in Fig. 3.1.

STEP 1: Modality-aware latent structure learning. For each modality m, it constructs an initial k-nn graph; using the raw features for each modality.

1. First, it computes the similarity between two items of the same modality. Let n be the number of items; the similarity matrix $S^m \in \mathbb{R}^{n \times n}$ for modality m, is computed from the vectors for each item e_i^m as

$$_{ij}^{m} = \frac{(e_i^m)^\top e_j^m}{||e_i^m|| ||e_j^m||}.$$

2. Next, it conducts a kNN sparsification, both to have only positive values and to maintain the most important values:

$$\hat{S}_{ij}^m = \begin{cases} S_{ij}^m, \text{ if } S_{ij}^m \in top-k(S_i^m) \\ 0, \text{ otherwise} \end{cases}.$$

This provides a sparse and oriented graph adjacency matrix.

3. Then, it normalizes the matrix with the diagonal degree matrix of $\hat{S}^m$, denoted as D^m, that responds to the sum of each row of $\hat{S}^m$.

$$\tilde{S}^m = (D^m)^{-\frac{1}{2}} \hat{S}^m (D^m)^{-\frac{1}{2}}.$$

   We want to acknowledge that, given the Laplacian (L) matrix (of a non directed graph), the normalized Laplacian (see Def. 2.55) is computed as $\mathcal{L} = (D)^{-\frac{1}{2}} L (D)^{-\frac{1}{2}}$. In this particular case, since the matrix is not symmetrical because the graph is not directed, this operation can look uncommon. The normalized Laplacian for a directed graph, called Diplacian, can defined in terms of probability matrices and distributions for irreducible Markov chains (see [16, Definition 3.2]). What is computed in this step is not the normalized Laplacian matrix, nor the Diplacian matrix, but it still converges.

4. To remove the noise and complete the information in the raw multi-modal features, it transforms raw modality features into high-level features by using a matrix and a vector of trainable parameters. They are initialized according to the values obtained in this study [8], known as Xavier Uniform, and they are modified when the model learns.

$$\tilde{e}_i^m = W_m e_i^m + b_m.$$

   It repeats the process used for the initial vectors and constructs $\tilde{A}^m$.

5. Lastly, it adds a skip connection to combine the learned graph with the initial graph, using a parameter $\lambda \in (0, 1)$

$$A^m = \lambda \tilde{S}^m + (1 - \lambda) \tilde{A}^m.$$

STEP 2: Aggregate multimodal latent graphs. Aggregates both textual and image graphs. It does by using two coefficients α_t and α_v that satisfy $\alpha_t + \alpha_v = 1$

$$A = \alpha_t A^t + \alpha_v A^v.$$

STEP 3: Graph convolutions. As we have already seen in other models, this step allows to introduce information from neighbouring nodes into the current one. To do so, it uses many convolution layers. Let $h_i^{(0)} \in \mathbb{R}^d$ be the embedding vector, $\mathcal{N}(i)$ the neighbor items. The l-th layer computes

$$h_i^{(l)} = \sum_{j \in \mathcal{N}(i)} A_{ij} h_j^{l-1}.$$

After L layers, $h_i^{(L)} \in \mathbb{R}^d$ encodes high-order item-item relationship, and can be used to improve CF methods.

STEP 4: Combining with collaborative filtering. Having chosen a collaborative filtering strategy, and calling $\tilde{x}_u, \tilde{x}_i \in \mathbb{R}^d$ the output embedding vectors for users and vectors in the CF method; we can combine the normalized item embeddings $h_i^{(L)}$ with the expression

$$\hat{x}_i = \tilde{x}_i + \frac{h_i^{(L)}}{||h_i^{(L)}||_2}$$

and compute the user-item preference score as usual

$$\hat{y}_{ui} = \tilde{x}_u^{\mathsf{T}} \hat{x}_i.$$

The fact that it can be easily incorporated into any CF method, makes it a play-and-plug complement to each of them.

STEP 5: Optimization. It uses Bayesian Personalized Ranking [23] to compute pair-wise ranking, and, in doing that, it encourages the prediction of an observed entry to be higher than its unobserved counterparts

$$\mathcal{L}_{BPR} = -\sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}_u} \sum_{j \notin \mathcal{I}_u} ln\sigma(\hat{y}_{ui} - \hat{y}_{uj})$$

where $\mathcal{I}_u$ indicates the observed items associated with the user u and (u, i, j) denotes the pairwise training triples where $i \in \mathcal{I}_u$ is the positive item and $j \notin \mathcal{I}_u$ is the negative item sampled from unobserved interactions. σ is the sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}.$$

It iterates this steps until the Recall@20 does not increase for 10 successive epochs, to avoid overfitting.

Chapter 3

Analysis, design and implementation

This chapter focuses on the design choices made and the code implementation. First, it takes a look at all the models studied in previous chapters and discusses how TDA could be incorporated into them. It also takes performance into account, and explains our choice to modify LATTICE. The second part of the chapter details how the code is organized, and which techniques and libraries are used. Lastly, it concludes with a detailed explanation on how TDA has been incorporated into the Recommender System.

3.1 Model choice

In this section, we discuss the models studied in Section 2.2.2, and how suitable is TDA to be incorporated into each of them. Then, we make a decision on which model to modify.

We want to note that all the models discussed are multimodal, and take the same information from product images and descriptors. Since the preprocessing for those is identical on each model, we can easily incorporate TDA to extract textual and image features in all of them. When we look at the implementation, the models that construct an graph are specially interesting, since TDA can extract descriptors from those, and it could be incorporated into the models. In particular, LATTICE computes a graph for image similarity, another for textual similarity and it combines them into a multimodal graph, which makes it specially suitable. These graphs evolve with the model learning, and this allows LATTICE to learn latent structures. Lastly, we looked at all the models studied, we compared the performance of all of them (see Tab. 4.3) and saw that LATTICE was the clear

winner. Weighting all these reasons, we have decided to incorporate TDA into LATTICE and the details on how we have done it are detailed on Sections 3.2, 3.3.

3.2 Code organization

The code for this project is organized in two git repositories:

- TDA-Notebooks [1]: Contains `Jupyter Notebooks` with `Python` code. They detail how TDA is performed on each step of the process, and allow to visually see the results. There are three of them, one for each structure where we apply TDA:
  - **Directed graph TDA:** first, it performs an investigation on the graph to check that it is a directed graph and showcase the weight range both for nodes and edges, as well as how connected they are. Later, it computes the Flagser persistence (see Def. 2.27) of the graph, using the `GiottoTDA` library [26]. Next, it computes all numeric and vectorial descriptors, as well as the areas under those. Lastly, it removes 25% of the nodes in the graph randomly, 10 times. It displays the persistence diagram of the operation that modifies the persistence values less.
  - **Image TDA:** it experiments with the grayscale image, by turning it into a cubical complex using the `Gudhi` library [27]. Using the same library, it computes all the descriptors for the image retrieved. Lastly, it sets up a pipeline that, given an image link, convolves and binarizes the grayscale image (to ease feature extraction and reduce noise). Then, it computes all the descriptors for the three versions of the image.
  - **Text TDA:** first looks into how many items have a description, and the average length for them. Later, it enriches them by concatenating brand, title and categories into it. With this, it has sentences that are long enough to extract descriptors. It uses Word2Vec [21]: a natural language processing technique that represents words as vectors in a continuous vector space, and captures semantic relationships and similarities between words. With Word2Vec, it obtains embeddings for each word, and computes, for each item, a dissimilarity matrix based on the word similarity provided by the Word2Vec library, that is cosine similarity between vectors. With this, it can use Vietoris Rips (see Def. 2.24) to compute and display the persistence and all the derived descriptors. It also divides each enriched description into chunks, adds the vectors

[1] Can be found at https://github.com/EstherRH00/TDA-Notebooks

representing the words on each chunk (as suggested in [1]), and computes the similarity between those. Even with overlapping chunks, the information retrieved is quite small, so we do not use this method on future steps. Lastly, it computes the TF-IDF matrix for each row, but since the sentences are short, the values obtained are not as useful as the ones provided by Word2Vec, so we discard this approach too.

- TDA-LATTICE [2]: Contains the code that generates the results in the next chapter. It is heavily influenced by the resouces published in these papers: [32], [33] and [34], to get the base implementations for the studied models. With those, we modified the LATTICE implementation from the code referenced on the paper [32], and adapted it to compute TDA on specific steps of the project. Thanks to the notebooks that we have mentioned on the previous item, integrating TDA on each phase of the code was easy to do. It also contains an excel file with all the results.

3.3 LATTICE with TDA

The main goal of this work is to introduce tDA within LATTICE. We use the schema in Fig. 3.1, that sums up how LATTICE works, to see how can we introduce TDA into its workflow. We use LightGCN as the base Collaborative Filtering model to which we plug LATTICE into, following what is done in the paper [32], and the fact that is the best performer across the three datasets (see Tab. 4.3). To keep the LATTICE architecture as pristine as possible, we look into the structure and find two places that could benefit from TDA.

The first of them is the preprocessing of the data, indicated in Fig. 3.1 with an "A". We can use topological descriptors extracted from images as well as from text, and combine them with the features that LATTICE already works with, via concatenation. Another place that would not modify much the architecture is marked with a "B" in Fig. 3.1, and it is the point where the multi-modal graph is finally computed, before convolution. We can use TDA to extract information about this graph; but we have to take into account that this process is iterated, so either we compute descriptors one time with the initial graph and use them until the last results are obtained, or we compute new descriptors each time that the matrix W_m and the bias vector b in the *Latent structure learning* part of the process (Step 1) evolve. We have implemented both of them. We also use the topological information in the graph both to combine it with the item information that the neural network already has, via a new linear layer and also to prune the graph

[2]Can be found at https://github.com/EstherRH00/TDA-LATTICE

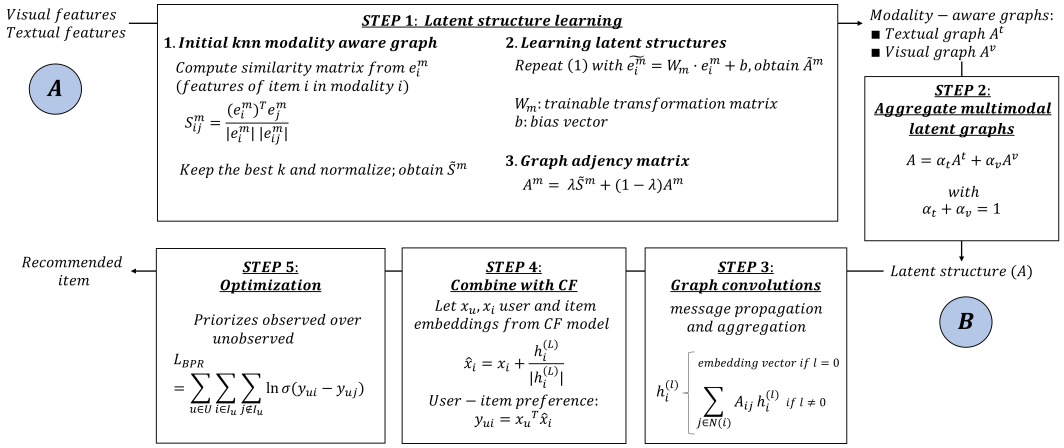


Figure 3.1: LATTICE workflow

and aim for better run times. This are detailed in Section 3.3.1, 3.3.2, 3.3.3.

3.3.1 Extraction of visual features

It corresponds to "A" in Fig. 3.1. We want to extract visual features with TDA and introduce them to the RS. LATTICE works with image features extracted by a CNN [10]; those are vectors of length 4096 and are pre-computed. On the other hand, topology can be used to describe images: we look at the gray-scale image as a Cubical complex (see Def. 2.18), where the value of each pixel represents the value of the voxel. Then, we use an algorithm to compute values for both the edges and the vertices in the complex. Given that, we use tresholding to obtain a filtration and compute homology. First we have computed it on a the image notebook (see Section 3.2) to test and visualize the code. Afterwards, we have incorporated it into the LATTICE code that computes features.

We consider all the descriptors that have been mentioned before, all of them have been done using the `Python` library [27]

1. **Numeric descriptors:** we consider total persistence, average lifetime, standard deviation and entropy (see Defs. 2.33, 2.34, 2.35 and 2.36).

2. **Vector descriptors:** we consider the Betti curve (see Def. 2.38). It is a continuous function, so we discretize it into 100 points and use these values. We use the vector obtained as a descriptor. For the landscapes (see Def. 2.40), we compute 10 landscapes and, again, we discretize it into 100 points. Then, we compute the area below each landscape and concatenate them into a vector, which is the descriptor that we use. Lastly, we compute the silhouettes (see Def. 2.41); and we use two different weight functions; the first one is $w_i = 1$

for each i, and the second one is $w_i = (d_i - b_i)^2$, from a persistence diagram $\{(b_i, d_i)\}_{i \in I}$. We use the area below these landscapes as the descriptor.

We compute the descriptors for the gray-scale image, a convoluted version of it and a binarized version of it, and use them all together by concatenating them. Lastly, we concatenate initial descriptors with the topological ones and feed the new vectors into LATTICE without further modification.

3.3.2 Extraction of text features

LATTICE also uses textual features from the descriptions of the items provided by the sellers. It does so by concatenating the information provided by the seller: categories, title, brand and description. Then, it passes this concatenated information to a trained BERT (Bidirectional Encoder Representations from Transformers) [22]. It captures contextualized representations of words in a way that considers both their left and right contexts, and generates a vector of 1024 positions for each item, that encapsulates the information captured.

In this work, we also propose to extract features with TDA in "A" in Fig. 3.1. We can refer to [7] to see how to use TDA to extract features from text: ther are two approaches to it; considering word embeddings or computing TF-IDF vectors for each word. Keeping in mind that we want to compute TDA descriptors for each product, we take a look into the shape of the descriptions, and, using the notebook described in Section 3.2, we observe that 40% of the descriptions are empty. Since a minimum text is necessary to extract features, we concatenate the title, brand, categories and description, and with this we had all of them not being empty and 95% of them containing more than 3 sentences.

Even with this improvement, we still faced two problems: the first one being that the sentences were short, so TF-IDF was giving small vectors with not much information. The second problem was that TF-IDF was only looking into structural similarity, contrarily to what LATTICE does. This means that, if we use TF-IDF in this two sentences: A B A B C and X Y X Y Z, we obtain the same graphs and TDA sees that they are identical, without taking into account differences in meaning.

To fix these two problems, we decided to use Word2Vec [21], in order to have vectors for each word that contain semantic information. We have experimented with two systems to build a graph for each item: on the first one, we consider each word as a vector and, on the second one, we build chunks with or without overlap from each sentence and add the vectors in each chunk together; following the paper [1]. With this, compute TDA in each of the graphs, considering the cosine similarity between pairs of vectors.

In the notebook, we see that the version without chunks retains more informa-

tion, so we compute the topological descriptors for that using the `Python` library Gudhi [27]. Similarly to what we did in Section 3.3.1, we concatenate descriptors obtained with BERT [22] with the topological descriptors obtained, and feed it into LATTICE.

3.3.3 Item graph

The second way that we have studied and analyzed to incorporate TDA in LATTICE is in the item graph, denoted as "B" in Fig. 3.1. The item graph gets modified in each epoch of learning and, acknowledging that computing TDA for a graph can take up to 30 seconds, we have to use it mindfully.

We have used the notebook described in Section 3.2 to study directed graphs and see how to compute TDA on an oriented graph using the `Giotto` library [26]. Then, we see that there are two ways to use it:

1. To add information into the learning process; we have tried it in two ways:
   (a) Computing TDA with an initial graph, as described on the Directed Graph Notebook, in Section 3.2. We have computed TDA separately for the original textual and image graph, and concatenated the descriptors. Then, on each learning epoch, we concatenate the TDA descriptors to each item descriptor and add a linear layer to reshape it into the original length. With training, this layer learns how to weight each of the original item descriptors and the TDA descriptors of the original graph, to use the information that improves the results more.
   (b) Computing TDA for the graph on each iteration. With the evolution of the item graph, we compute the same descriptors that we have computed for the original graph and, in a similar way, we concatenate them with the item descriptors that LATTICE already obtains. A linear layer reshapes it into the original length, and weights each of the descriptors.
2. To prune the graph. We study topologically the original graph, then drop some of the nodes that make the descriptors change less. To do so, we proceed like we have done on the Directed Graph Notebook, in Section 3.2. By removing nodes that alter the persistence properties less, we aim to make it faster (since the graph is smaller) and, hopefully, not loose much information (since the persistence properties are only slightly modified).

Chapter 4

Experiments

In this chapter we pose the research questions that we want to answer. Then, we describe the setup for the experiments: datasets used and baselines. Next, we define an evaluation protocol and we utilize implementations of the models mentioned in the reference paper [32] that were accessible to us (Lattice [32], MF, NGCF, LightGCN [33] and VBPR, MMGCN, GRCN [34]). We run all the codes, with slight modifications to adapt them to our architecture and to strongly fix the random seeds. Lastly, we compare the results between them and with the values the original paper obtained for the Baby dataset. We also look at which models performed best for each dataset.

4.1 Research questions

In this section, we define two research questions that motivate our experiments:

1. **RQ1: How does LATTICE perform compared with the state-of-the-art methods?** To answer this question, we compare the results obtained by the other six Recommender Systems against LATTICE. The mesures used to compare their performance are detailed on Section 4.3. The results are portrayed on Section 4.4
2. **RQ2: How does LATTICE with TDA perform compared to LATTICE?** To answer this question, we have run LATTICE and all the modifications described in the previous Chapter 3. The results are portrayed on Section 4.5.

4.2 Experimental settings

In this section, we detail the characteristics of the datasets used, and explain the baselines that we use to compare their performance.

Dataset	#Users	#Items	#Interactions	Density
Baby	19,445	7,050	160,792	0.00117
Digital Music	5,541	3,568	64,706	0.00327
Musical Instruments	1,429	900	10,261	0.00797

Table 4.1: **Dataset description:** statistics of the datasets used to evaluate the Recommender Systems.

We used the three smallest datasets available on [10] that have image information. This choice is encouraged by the fact that those are the small, which is usually a strength for TDA, and also because the Baby dataset is used on the LATTICE paper [32]; which allows us to compare the values obtained in that paper with our own, and verify that the implementations used are correct. The characteristics of each dataset are portrayed on Tab. 4.1, where Density $= \frac{\#\text{Interactions}}{\#\text{Users}\cdot\#\text{Items}}$.

Now we explain in detail the evaluation metrics used. To compare the performance of the different Recommender Systems we look at the 20 items with best affinity for a user u (relevant); and compare it with the 20 that each model has considered best for them (recommended). We evaluate each method with the following criteria:

- **Precision at 20 (P@20):** looks at the proportion of items in the top-20 set that are relevant.

$$P_{20} = \frac{\#\text{ of recommended items @20 that are relevant}}{20}.$$

- **Recall at 20 (R@20):** looks at the proportion of relevant items found in the top-20 recommendations.

$$R_{20} = \frac{\#\text{ of recommended items @20 that are relevant}}{\text{total \# of relevant items}}.$$

- **Normalized Discounted Cumulative Gain (NDCG@20):** similar to precision, but penalizes according to the relevance of each document. It is the quotient of two expressions:

  - **Discounted Cumulative Gain (DCG@20):** computes the quotient between the relevance of the document in position i and the position

$$CG_{20} = \sum_{i=1}^{20} \frac{\text{rel}_i}{\log_2(i+1)}.$$

– **Ideal Discounted Cumulative Gain (IDCG@20):** similar to DGC, but only looks into the most 20 relevant documents, listed in REL_{20}. It is equal to the maximum DCG possible

$$IDCG_{20} = \sum_{i=1}^{|\text{REL}_{20}|} \frac{\text{rel}_i}{\log_2(i+1)}.$$

We use the Ideal CDG to normalize CDG and obtain the expression

$$NDCG_{20} = \frac{DCG_{20}}{IDCG_{20}}.$$

The information it provides is usually seen as a substitute for Precission, since it rewards relevant items in the top ranked results than those ranked lower. However, we decide to keep both of them, in order to align with the standards followed on the original paper [32].

4.3 Evaluation protocol

It is important to note that, even if we fix the `Random`, `NumPy` and `PyTorch` seeds, results might still vary slightly. This is why, in order to compare performance, we run the code 5 times for each model and dataset; and compute the mean and variance values obtained for each of the indicators described above. We consider that there is an improvement or lose if the intervals mean$\pm$var are disjoint. If they overlap, we consider that the difference is not significative. We also have set the limit of iterations to 1000, but in all the cases it has converged earlier. We have structured the experiment in the following six steps:

1. Pre-process the data for the 3 datasets (computing the descriptors used in the paper and the topological ones).
2. Run all the studied models, except for LATTICE.
3. Fine-tune LATTICE with the parameters defined by the authors in [32, Section 3.1.4]: the learning rate is tuned amongst {0.0001, 0.0005, 0.001, 0.005} and the coefficient of normalization is searched in {0, 10^{-3}, 10^{-4}, 10^{-5}}. We compare the values obtained here with the ones in the paper, and fix one for the rest of the experiments involving LATTICE.
4. Run LATTICE with topological descriptors.
5. Incorporate LATTICE in the graph in all the ways listed before, one at a time.
6. Combine the best version of the pre-processing with the best version of the process for LATTICE.

4.4 Model comparison

First we run each model for each dataset and, afterwards, we display all the results in a table. The results are presented with 4 decimal positions, like they are in the paper [32], the full information can be found on the github of the project (see Section 3.2), that includes an excel file with all the results. We have modified the code slightly to moderate the variance between iterations by fixing the random, numpy and torch seeds. In the following pages, there will be a table for each model and dataset, fine-tuning for LATTICE and lastly a global comparison one. We denote P@20, R@20 and NDCG@20 the Precission at 20, Recall at 20 and Normalized Discounted Cumulative Gain at 20, respectively.

Looking at this first three models (see Tabs. 1, 2 and 3), we observe that LightGCN has the best performance of them, followed by NGCF and lastly MF, which is the worst performing but also the conceptually easier to understand. This ranking is true in all the datasets, but in the Baby dataset the difference is more notable: R@20 improves a 16% compared to NGCF and a 50% against MF. We also want to note that the amount of epochs required for each model varies and LightGCN is the model that requires less iterations for Musical Instruments and Baby, but it is the one that requires the most for Digital Music. NGCF requires twice as much epochs as MF for the Musical Instruments datasets and Baby, while it requires a 25% less than MF for Digital Music. With this first contact, we see that the Digital Music dataset seems to behave quite differently than the other two.

Comparing this second block of models (see Tabs. 4, 5 and 6), we observe that VPR is the worst performing one across all datasets. GRCN performs best on Digital Music and Baby, while MMGCN is the best performing for Musical Instruments. When benchmarking the second block against the the first one (see Tabs. 1, 2 and 3), we see that the performance of each model depends a lot on the dataset that it is being computed in. In all of them VBPR turns out the worst, but GRCN is slightly worse than LightGCN for the Musical Instruments and Digital Music datasets, while is the best performing one for the Baby dataset. Lastly, MMGCN performs the best for Musical Instruments, while it gives worst results than MF for Digital Music and, it falls between LightGCN and NGCF for the Baby data.

In summary, we have observed that LightGCN gives an overall good performance, ranking first for Digital Music and second for Baby and Musical Instruments. This is why it is the one that we use in LATTICE, just like the original paper [32] does.

	Learning rate	Normalization coef.
Baby	0.001	0.0
Digital Music	0.001	0.0
Musical Instruments	0.005	10^{-5}
Used in the paper [32]	0.0005	0

Table 4.2: **LATTICE fine-tuning:** optimal values for LATTICE in each dataset

4.4.1 LATTICE

Before running Lattice, we have fine-tuned the model with the parameter values used in the paper [32, Section 3.1.4]: the learning rate amongst {0.0001, 0.0005, 0.001, 0.005} and the coefficient of normalization in {0, 10^{-3}, 10^{-4}, 10^{-5}}. For each dataset we obtained different values; shown in Tab. 4.2.

It is worth mentioning that the paper where LATTICE is described includes an evaluation with Clothing, Sports and Baby datasets. We have opted for smaller datasets, since TDA usually performs better on those. Taking into account that the datasets used in the paper are substantially larger than the ones we used, we find this discrepancies normal. From now on, and in order to be consistent with the paper, we use 0.0005 as the learning rate and 0 as the normalization coefficient. We now run LATTICE five times with this values (see Tab. 7), just like we did with the other models, and compare the results on Tab. 4.3.

To finish up this section, in Tab. 4.3, we compare the performance obtained by the baselines and LATTICE on each dataset. We compare the columns on Tab. 4.3 corresponding to the Baby dataset with the values found on the paper [32], and it allows us to confirm the models are correctly implemented, since we obtain really close results. We also see that LATTICE outperforms all the other models in all the metrics that we are evaluating, and this answers the Research Question 1.

4.5 LATTICE updated with TDA

For this section, we first evaluate each of the three possible pre-processings, denoted with "A" in Fig. 3.1 (see Section 4.5.1). Next, in Section 4.5.2, we compare three modifications for LATTICE Neural Network, denoted with "B" in Fig. 3.1. Lastly, we combine the best performing option for each step.

	Musical instruments			Digital music			Baby		
Algorithms	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20
MF	0.1115	0.0058	0.0478	0.2315	0.0144	0.1238	0.0457	0.0024	0.0202
NGCF	0.1427	0.0074	0.0641	0.2573	0.0159	0.1354	0.0591	0.0031	0.0254
LightGCN	0.1598	0.0083	0.0702	*0.2904*	*0.0179*	*0.1561*	0.0687	0.0036	0.0306
GRCN	0.1533	0.0079	0.0621	0.2864	0.0179	0.1413	*0.0802*	*0.0042*	*0.0346*
MMGCN	*0.1857*	*0.0095*	*0.0798*	0.2051	0.0127	0.0909	0.0608	0.0032	0.0148
VBPR	0.1047	0.0054	0.0448	0.1186	0.0075	0.0507	0.0286	0.0015	0.0121
LATTICE	**0.2090**	**0.0107**	**0.0925**	**0.2929**	**0.0180**	**0.1544**	**0.0829**	**0.0044**	**0.0366**

Table 4.3: **Model comparison.** Overall performance comparison of LATTICE with different baselines, in terms of Recall (R@20), Precission (P@20) and NDCG@20. We mark with **bold** the best model for each dataset, and with *italic* the second one.

4.5.1 LATTICE with TDA in "A"

First of all, when adding TDA into the image pre-processing (see "A" in Fig. 3.1), we see an improvement of 2.6% average in the smallest dataset, Musical Instruments, (1.85% in recall, 2.55% in precision and 3.60% in NDCG), while for Digital Music the recall increments in 0.3% and the NDCG in 0.04%, but precision looses 0.12%. For the largest dataset, Baby, we see a loss of 0.90% on average against LATTICE. In all of this situations, the intervals mean$\pm$var are disjoint; so we consider that both the improvements and losses are significant (see Tab. 8).

On the other hand, when adding TDA into the text pre-processing (see Tab. 9), we see that all the metrics fall, a 1.32%, 0.72%, and 0.75% average for Musical Instruments, Digital Music and Baby respectively against LATTICE. This losses are significant, since the intervals mean$\pm$var are disjoint in all cases.

Lastly, when combining both types of preprocessing and comparing it against LATTICe, we observe the following (see Tab. 10): for Musical Instruments, the recall and precision have deteriorated a 1.16% and 1.24% respectively, but NDCG increases by a 0.01%. In Digital Music, we see an improvement of 0.47% on average through all the fields (0.51% in recall, 0.46% in precision and 0.43% in NDCG) For the Baby dataset, all of them drop, on average a 7.5%. All the losses reported have been computed by taking the average recall, precision and NDCG in the original LATTICE implementation (see Tab. 7, subtracting the new value obtained and calculating the quotient between this difference and the original LATTICE value.

4.5.2 LATTICE with TDA in "B"

For each of the following modifications, we use the original textual and image descriptors. We have decided to compute TDA on "B" because there are two different directed weighted graphs, one for image features and one for textual features, and TDA is suitable for extracting information of them and, as described in Section 3.3.3, use it to prune the graph or feed it into the NN.

First, we compute TDA of the initial graph and on each epoch we concatenate it with the item descriptors obtained, to later pass it through a neural layer that turns it into the original shape. Looking at the results seen in Tab. 11, we see a notable deterioration on the results in all datasets: Musical Instruments, Digital Music and Baby drop a 13.55%, 25% and 24.5% respectively. We consider that this happens because of the complex LATTICE architecture: a small perturbation in the neuron layers has a great impact on the result. Furthermore, by adding a layer, we are making the model deeper, which is not advised for Recommender Systems.

Afterwards, we do the same but computing TDA on each epoch (see Tab. 12). The results are, on average, 12.35% worse for the Musical Instruments dataset, 25.89% worse for the Digital Music and 25.49% for the Baby. On top of that, and the execution time increases between a 116% and up to 250%. Our conclusions about introducing TDA within the item graph are that it strongly affects the performance and it causes it to drop. We consider that the complex architecture is very vulnerable to small changes, and that when modifying the item graph probably other changes have to be done. We hypothesize that revisiting the parameters and fine-tuning again could improve the performance a bit.

Lastly, using TDA to prune the original graph (see Tabs. 14, 13 and 15), we see that the results drop; even by only removing 4% of the nodes. The best we accomplish is a loss of 86.92% on average for the Musical Instruments dataset, 97.81% for Digital Music, and 94% for Baby. While the execution time goes down by 50.91% and 65.49% on the two first datasets, it increases by over 256% on the Baby one, compared to the original LATTICE.

Lastly, we compare all the modifications against the original LATTICE implementation (see Tab. 4.4) and observe that the performance has decreased with all the modifications that we have performed on the processing (see "B" on Fig. 3.1), and the only notable improvements are the ones made with TDA on images, when applied to small datasets. Specifically, Musical Instruments benefits from the changes in the image preprocessing by 2.5%. Digital music sees improvements of less than 1.5% in this implementation. When combined with the text preprocessing, we see a slight increment in performance against the original LATTICE in some of the evaluation metrics for Musical instruments and Digital Music. All the other modifications have implied a notable loss on the results.

	Musical instruments			Digital music			Baby		
Algorithm	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20
Original LATTICE [32]	0.2090	0.0107	0.0925	0.2929	0.0180	0.1544	0.0825	0.0043	0.0365
TDA Image	*0.2129*	**0.0110**	**0.0959**	0.2938	0.0180	0.1544	0.0819	0.0043	0.0360
TDA Text	0.2060	0.0105	0.0917	0.2907	0.0179	0.1536	0.0765	0.0040	0.0335
TDA Image and text	0.2066	0.0106	0.0925	0.2945	0.0181	0.1550	0.0764	0.0040	0.0335
TDA First graph	0.1796	0.0093	0.0805	0.2223	0.0138	0.1118	0.0623	0.0033	0.0274
TDA each graph	0.1836	0.0094	0.0805	0.2208	0.0137	0.1111	0.0614	0.0032	0.0271
TDA drop 1/25	0.0294	0.0015	0.0106	0.0066	0.0004	0.0026	0.004	0.0002	0.0012
TDA drop 1/20	0.0202	0.0010	0.0073	0.0056	0.0003	0.0022	0.0038	0.0002	0.0016
TDA drop 1/15	0.0241	0.0012	0.0094	0.0061	0.0004	0.0028	0.0053	0.0003	0.0017
TDA drop 1/10	0.0275	0.0014	0.0094	0.0066	0.0004	0.0030	0.0035	0.0002	0.0013
TDA drop 1/5	0.0246	0.0013	0.0092	0.0059	0.0004	0.0031	0.0038	0.0002	0.0012

Table 4.4: **Model comparison.** Overall performance comparison of LATTICE against different modifications , in terms of Recall (R@20), Precission (P@20) and NDCG@20. We mark with **bold** improvements of over 2.5%, and with *italic* improvements less than 2% and over 1.5% and underline other improvements

We conclude, seeing the results in Tab. 4.4, that TDA has been useful to extract information from images. In this field, it is known for being able to extract data about textures and it has benefited the model for the Musical Instruments dataset. When extracting data from textual descriptors, it has not been fruitful. TDA is often used to find structure in the text, and the textual data that we had was small, so we hypothesize that there was not much information to retrieve about the structure of the argument. It also seems to mean that the user-item interactions are not strongly related to the schema of the textual description given for the products. Lastly, all changes made in the LATTICE architecture have caused losses. We attribute this to the fact that these changes have not been accompanied with a change in any other of the parameters, and modifying a single step in a complex architecture, like the one LATTICE presents, has notably big consequences. Furthermore, when we have added a new layer, we have made the network deeper, which is usually not advised. Finally, when pruning the graph the results have been catastrophic, and from that we deduce that LATTICE hardly relys on the information it has, and that even a small reduction on that amount has a great impact on the performance. With all this, we can answer the Research Question 2 and say that the modifications perform poorly in most cases, with exceptions when additional information about the images is provided, in small datasets.

Chapter 5

Conclusions and future work

In this chapter, we discuss the work done and future improvements that could be conducted. We consider that we have covered the bases of Topological Data Analysis, with a strong mathematical foundation that we have used in many ways throughout the report. With these methods, and a profound study on Recommender Systems, we have managed to find one that could benefit from Topological Data, and have implemented six different ways to incorporate it, three of them add more information to the product information, before starting with the learning process. The other three, alter the process with two different objectives: to infuse topological information in the neural layers and to modify the graph preserving its topological properties.

We have computed topological descriptors for three different types of data: directed graphs, point clouds and images. We have also realised that modifying a complex architecture comes with many problems, and changing shapes or adding layers can deeply change the results it produces.

With this, we consider that we have accomplished our objectives, since we have successfully used topology to modify and, in some cases improve, the quality of the recommendations provided by a multimodal collaborative filtering system. It is also worth mentioning that to the best of our knowdledge this is the first attempt to include TDA in this kind of Recommendation Systems. So, all the proposals presented in this work are novel. Our results denote some improvements and deteriorations of the original results. In any case, many lessons learned have appeared:

- Topological descriptors obtained from images are useful to retain information about textures.
- Topological descriptors obtained from text are useful to understand the text structure.

- The Recommender System LATTICE, performs better in small datasets when more information about images is provided.
- LATTICE has a complex architecture and modifying it produces great perturbations on the results.
- The information about items is crucial for LATTICE, and when removing it the results fall.

This work could be extended by incorporating new datasets and adjusting the metrics computed, such as the number of descriptors. Exploring the incorporation of Topological Data Analysis into the graph generated from a cold start is particularly intriguing, as we hypothesize that topological data could effectively fill in gaps. Moreover, assigning greater weight to TDA descriptors in the computation of feature vectors might result in more significant variations in the results.

Bibliography

[1] E. Amigó, A. Ariza-Casabona, V. Fresno, and M. A. Martí. Information theory-based compositional distributional semantics. Computational Linguistics, 48(4):907–948, 12 2022.

[2] B. Bleile, A. Garin, T. Heiss, K. Maggs, and V. Robins. The Persistent Homology of Dual Digital Image Constructions, pages 1–26. 01 2022.

[3] G. Carlsson. Topology and data. Bulletin of The American Mathematical Society - BULL AMER MATH SOC, 46:255–308, 04 2009.

[4] F. Chazal, D. Cohen-Steiner, M. Glisse, L. Guibas, and S. Oudot. Proximity of persistence modules and their diagrams. Proc. 25th ACM Sympos. Comput. Geom., pages 237–246, 12 2008.

[5] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. In Proceedings 41st Annual Symposium on Foundations of Computer Science, pages 454–463, 2000.

[6] H. Freudenthal. Simplizialzerlegungen von beschränkter Flachheit. Annals of Mathematics, 43(3):580–582, 1942.

[7] S. Gholizadeh, K. Savle, A. Seyeditabari, and W. Zadrozny. Topological data analysis in text classification: Extracting features with additive information, 2020.

[8] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. Journal of Machine Learning Research - Proceedings Track, 9:249–256, 01 2010.

[9] A. Hatcher. Algebraic topology. Cambridge Univ. Press, Cambridge, 2000.

[10] R. He and J. McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In Proceedings of the 25th International Conference on World Wide Web, WWW '16. International World Wide Web Conferences Steering Committee, Apr. 2016.

[11] R. He and J. McAuley. Vbpr: Visual bayesian personalized ranking from implicit feedback. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, pages 144–150. AAAI Press, 2016.

[12] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '20, pages 639–648, New York, NY, USA, 2020. Association for Computing Machinery.

[13] J. Hu, Y. Liu, J. Zhao, and Q. Jin. MMGCN: Multimodal fusion via deep graph convolution network for emotion recognition in conversation. In C. Zong, F. Xia, W. Li, and R. Navigli, editors, Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 5666–5675, Online, Aug. 2021. Association for Computational Linguistics.

[14] T. Kaczynski, K. M. Mischaikow, and M. Mrozek. Computational homology. Springer, 2004.

[15] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. Computer, 42(8):30–37, 2009.

[16] Y. Li and Z.-L. Zhang. Digraph laplacian and the degree of asymmetry. Internet Mathematics, 8, 12 2012.

[17] D. Lütgehetmann, D. Govc, J. P. Smith, and R. Levi. Computing persistent homology of directed flag complexes. Algorithms, 13(1), 2020.

[18] P. Melville and V. Sindhwani. Recommender systems. In C. Sammut and G. I. Webb, editors, Encyclopedia of Machine Learning and Data Mining, pages 1056–1066. Springer US, Boston, MA, 2017.

[19] P. Pilarczyk and P. Real. Computation of cubical homology, cohomology, and (co)homological operations via chain contraction. Adv. Comput. Math., 41(1):253–275, feb 2015.

[20] L. Polterovich, D. Rosen, K. Samvelyan, and J. Zhang. Topological persistence in geometry and analysis, 2021.

[21] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In

Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, pages 45–50, Valletta, Malta, May 2010. ELRA.

[22] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 11 2019.

[23] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI '09, pages 452–461, Arlington, Virginia, USA, 2009. AUAI Press.

[24] F. Ricci, L. Rokach, and B. Shapira, editors. Recommender Systems Handbook. Springer US, 2022.

[25] V. K. Singh, S. Sabharwal, and G. Gabrani. Comprehensive analysis of multimodal recommender systems. In I. Jeena Jacob, S. Kolandapalayam Shanmugam, S. Piramuthu, and P. Falkowski-Gilski, editors, Data Intelligence and Cognitive Informatics, pages 887–901, Singapore, 2021. Springer Singapore.

[26] G. Tauzin, U. Lupo, L. Tunstall, J. B. Pérez, M. Caorsi, A. Medina-Mardones, A. Dassatti, and K. Hess. giotto-tda: A topological data analysis toolkit for machine learning and data exploration, 2020.

[27] The GUDHI Project. GUDHI User and Reference Manual. GUDHI Editorial Board, 2015.

[28] H. Wagner, C. Chen, and E. Vuçini. Efficient computation of persistent homology for cubical data. In R. Peikert, H. Hauser, H. Carr, and R. Fuchs, editors, Topological Methods in Data Analysis and Visualization II: Theory, Algorithms, and Applications, pages 91–106. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[29] F. Wang, H. Wagner, and C. Chen. GPU Computation of the Euler Characteristic Curve for Imaging Data. In X. Goaoc and M. Kerber, editors, 38th International Symposium on Computational Geometry (SoCG 2022), volume 224 of Leibniz International Proceedings in Informatics (LIPIcs), pages 64:1–64:16, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[30] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua. Neural graph collaborative filtering. In Proceedings of the 42nd International ACM SIGIR Conference on

Research and Development in Information Retrieval, SIGIR'19, pages 165–174, New York, NY, USA, 2019. Association for Computing Machinery.

[31] Y. Wei, X. Wang, L. Nie, X. He, and T.-S. Chua. Graph-refined convolutional network for multimedia recommendation with implicit feedback. In Proceedings of the 28th ACM International Conference on Multimedia, MM '20, pages 3541–3549, New York, NY, USA, 2020. Association for Computing Machinery.

[32] J. Zhang, Y. Zhu, Q. Liu, S. Wu, S. Wang, and L. Wang. Mining latent structures for multimedia recommendation. In Proceedings of the 29th ACM International Conference on Multimedia, pages 3872–3880, 2021.

[33] J. Zhang, Y. Zhu, Q. Liu, M. Zhang, S. Wu, and L. Wang. Latent structure mining with contrastive modality fusion for multimedia recommendation. IEEE Transactions on Knowledge and Data Engineering, 35(9):9154–9167, 2023.

[34] X. Zhou, H. Zhou, Y. Liu, Z. Zeng, C. Miao, P. Wang, Y. You, and F. Jiang. Bootstrap latent representations for multi-modal recommendation. In Proceedings of the ACM Web Conference 2023, pages 845–854, 2023.

[35] A. Zomorodian and G. Carlsson. Computing persistent homology. Discrete and Computational Geometry, 33:249–274, 02 2005.

Result tables

This appendix incorporates the results of each execution performed. It also includes the computation of the mean and variance values for the five runs of each model, as well as temporal and epoch information when suitable.

.1 Recommender Systems

The results of MF, NGCF, LightGCN, GRCN, MMGCN and VBPR are portrayed in Tabs. 1, 2, 3, 4, 5 and 6, respectively. Lastly, the results of LATTICE can be found in Tab. 7.

.2 LATTICE with TDA in "A"

The results of modifications of LATTICE on the pre-processing steps, marked with an "A" in Fig. 3.1 are displayed in this section. The outcome obtained when incorporating TDA descriptors into image, text and both is displayed in Tabs. 8, 9 and 10, respectively. The values obtained on each of the five runs are shown, as well as the mean values, variance and the original LATTICE results, to ease the comparison.

.3 LATTICE with TDA in "B"

The results of modifications of LATTICE on the item graph, marked with a "B" in Fig. 3.1 are displayed in this section. The outcome obtained when computing TDA on the original graph and incorporating it to the NN and when computing TDA on the graph in each are shown in Tab. 11 and Tab. 12, respectively. The values obtained on each of the five runs are shown, as well as the mean values, variance and the original LATTICE results, to ease the comparison. Lastly, the values obtained when pruning 1/5, 1/10, 1/15, 1/20 and 1/25 of the nodes are

	Musical instruments			Digital music			Baby		
Iteration	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20
1	0.1115	0.0058	0.0478	0.2315	0.0144	0.1238	0.0457	0.0024	0.0202
2	0.1115	0.0058	0.0478	0.2315	0.0144	0.1238	0.0457	0.0024	0.0202
3	0.1115	0.0058	0.0478	0.2315	0.0144	0.1238	0.0457	0.0024	0.0202
4	0.1115	0.0058	0.0478	0.2315	0.0144	0.1238	0.0457	0.0024	0.0202
5	0.1115	0.0058	0.0478	0.2315	0.0144	0.1238	0.0457	0.0024	0.0202
Mean	0.1115	0.0058	0.0478	0.2315	0.0144	0.1238	0.0457	0.0024	0.0202
Var	0	0	0	0	0	0	0	0	0

Table 1: **Matrix factorization:** performance comparison of the Recommender System MF with different datasets.

	Musical instruments			Digital music			Baby		
Iteration	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20
1	0.1461	0.0075	0.0649	0.2572	0.0160	0.1347	0.0608	0.0032	0.0260
2	0.1421	0.0073	0.0635	0.2576	0.0160	0.1356	0.0600	0.0032	0.0254
3	0.1415	0.0073	0.0635	0.2574	0.0159	0.1353	0.0598	0.0032	0.0258
4	0.1462	0.0076	0.0660	0.2587	0.0160	0.1364	0.0580	0.0031	0.0251
5	0.1377	0.0071	0.0626	0.2557	0.0158	0.1350	0.0569	0.0030	0.0248
Mean	0.1427	0.0074	0.0641	0.2573	0.0159	0.1354	0.0591	0.0031	0.0254
Var	$1.3{\cdot}10^{-5}$	$3.3{\cdot}10^{-8}$	$1.7{\cdot}10^{-6}$	$1.1{\cdot}10^{-6}$	$6.1{\cdot}10^{-9}$	$4.2{\cdot}10^{-7}$	$2.6{\cdot}10^{-6}$	$6.7{\cdot}10^{-9}$	$2.5{\cdot}10^{-7}$

Table 2: **NGCF:** performance comparison of the Recommender System NGCF with different datasets.

	Musical instruments			Digital music			Baby		
Iteration	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20
1	0.1598	0.0083	0.0702	0.2904	0.0179	0.1561	0.0687	0.0036	0.0306
2	0.1598	0.0083	0.0702	0.2904	0.0179	0.1561	0.0687	0.0036	0.0306
3	0.1598	0.0083	0.0702	0.2904	0.0179	0.1561	0.0687	0.0036	0.0306
4	0.1598	0.0083	0.0702	0.2904	0.0179	0.1561	0.0687	0.0036	0.0306
5	0.1598	0.0083	0.0702	0.2904	0.0179	0.1561	0.0687	0.0036	0.0306
Mean	0.1598	0.0083	0.0702	0.2904	0.0179	0.1561	0.0687	0.0036	0.0306
Var		0	0	0	0	0	0	0	0

Table 3: **LightGCN:** performance comparison of the Recommender System MF with different datasets.

	Musical instruments			Digital music			Baby		
Iteration	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20
1	0.1528	0.0079	0.0616	0.2829	0.0178	0.1401	0.0801	0.0043	0.0349
2	0.1504	0.0077	0.0605	0.2896	0.018	0.1413	0.0802	0.0043	0.0345
3	0.1535	0.0079	0.0616	0.2865	0.0179	0.1417	0.0798	0.0042	0.0345
4	0.163	0.0084	0.0654	0.2854	0.0179	0.1403	0.0804	0.0043	0.0349
5	0.1472	0.0076	0.0616	0.2877	0.018	0.1429	0.0805	0.0043	0.0343
Mean	0.15338	0.0079	0.06214	0.2864	0.0179	0.1413	0.0802	0.00428	0.03462
Var	$3.5{\cdot}10^{-5}$	$9.5{\cdot}10^{-8}$	$3.6{\cdot}10^{-6}$	$6.3{\cdot}10^{-6}$	$7{\cdot}10^{-9}$	$1.3{\cdot}10^{-6}$	$7.5{\cdot}10^{-8}$	$2{\cdot}10^{-9}$	$7.2{\cdot}10^{-8}$

Table 4: **GRCN:** performance comparison of the Recommender System GRCN with different datasets.

	Musical instruments			Digital music			Baby		
Iteration	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20
1	0.1773	0.0091	0.0777	0.2076	0.0129	0.0922	0.0627	0.0033	0.0149
2	0.1869	0.0096	0.081	0.2052	0.0128	0.0921	0.0588	0.0031	0.0145
3	0.1866	0.0096	0.0789	0.2001	0.0124	0.0885	0.0624	0.0033	0.0154
4	0.1866	0.0096	0.0802	0.2127	0.0132	0.0952	0.0611	0.0033	0.0153
5	0.1915	0.0098	0.0813	0.2001	0.0125	0.0865	0.0593	0.0032	0.0142
Mean	0.1857	0.0095	0.0798	0.2051	0.0127	0.0909	0.0608	0.0032	0.0148
Var	$2.7{\cdot}10^{-5}$	$6.8{\cdot}10^{-8}$	$2.3{\cdot}10^{-6}$	$2.9{\cdot}10^{-5}$	$1{\cdot}10^{-7}$	$1.2{\cdot}10^{-5}$	$3.1{\cdot}10^{-6}$	$8{\cdot}10^{-9}$	$2.6{\cdot}10^{-7}$

Table 5: **MMGCN:** performance comparison of the Recommender System MMGCN with different datasets.

	Musical instruments			Digital music			Baby		
Iteration	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20
1	0.1047	0.0054	0.0448	0.1186	0.0075	0.0507	0.0286	0.0015	0.0121
2	0.1047	0.0054	0.0448	0.1186	0.0075	0.0507	0.0286	0.0015	0.0121
3	0.1047	0.0054	0.0448	0.1186	0.0075	0.0507	0.0286	0.0015	0.0121
4	0.1047	0.0054	0.0448	0.1186	0.0075	0.0507	0.0286	0.0015	0.0121
5	0.1047	0.0054	0.0448	0.1186	0.0075	0.0507	0.0286	0.0015	0.0121
Mean	0.1047	0.0054	0.0448	0.1186	0.0075	0.0507	0.0286	0.0015	0.0121
Var	0	0	0	0	0	0	0	0	0

Table 6: **VBPR:** performance comparison of the Recommender System VBPR with different datasets.

	Musical instruments			Digital music			Baby		
Iteration	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20
1	0.2105	0.0108	0.0934	0.2955	0.0182	0.1552	0.0825	0.0043	0.0366
2	0.2098	0.0107	0.0933	0.2899	0.0178	0.1525	0.0835	0.0044	0.0366
3	0.2098	0.0107	0.0927	0.2906	0.0178	0.1536	0.0828	0.0044	0.0366
4	0.2095	0.0108	0.0922	0.2956	0.0182	0.1547	0.0829	0.0044	0.0366
5	0.2054	0.0105	0.0910	0.2931	0.0180	0.1559	0.0827	0.0043	0.0366
Mean	0.2090	0.0107	0.0925	0.2929	0.0180	0.1544	0.0829	0.0044	0.0366
Var	$4.1{\cdot}10^{-6}$	$1.1{\cdot}10^{-8}$	$9.8{\cdot}10^{-7}$	$7.1{\cdot}10^{-6}$	$3.1{\cdot}10^{-8}$	$1.8{\cdot}10^{-6}$	$1.2{\cdot}10^{-10}$	$2.8{\cdot}10^{-10}$	$6.1{\cdot}10^{-10}$

Table 7: **LATTICE:** performance comparison of the Recommender System LATTICE with different datasets.

	Musical instruments			Digital music			Baby		
Iteration	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20
1	0.2161	0.0112	0.0972	0.2961	0.0181	0.1556	0.0821	0.0043	0.0361
2	0.2178	0.0112	0.0982	0.2958	0.0179	0.1539	0.0812	0.0043	0.0359
3	0.2094	0.0108	0.0945	0.2958	0.0181	0.1556	0.0823	0.0043	0.0361
4	0.2098	0.0108	0.0943	0.2928	0.0180	0.1539	0.0820	0.0043	0.0361
5	0.2115	0.0109	0.0951	0.2887	0.0178	0.1531	0.0816	0.0043	0.0360
Mean	0.2129	0.0110	0.0959	0.2938	0.0180	0.1544	0.0819	0.0043	0.0360
Var	$1.5{\cdot}10^{-5}$	$3.9{\cdot}10^{-8}$	$3{\cdot}10^{-6}$	$1{\cdot}10^{-5}$	$2.3{\cdot}10^{-8}$	$1.3{\cdot}10^{-6}$	$2.0{\cdot}10^{-7}$	$4.7{\cdot}10^{-10}$	$8.8{\cdot}10^{-9}$
LATTICE	0.2090	0.0107	0.0925	0.2929	0.0180	0.1544	0.0829	0.0044	0.0366

Table 8: **LATTICE with TDA image preprocessing:** performance of LATTICE with TDA with image preprocessing in different datasets. The LATTICE baseline is provided in the last line to facilitate ease of comparison.

	Musical instruments			Digital music			Baby		
Iteration	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20
1	0.2054	0.0105	0.0919	0.2929	0.0180	0.1551	0.0761	0.0040	0.0334
2	0.2047	0.0105	0.0911	0.2884	0.0177	0.1530	0.0779	0.0041	0.0339
3	0.2070	0.0106	0.0921	0.2889	0.0177	0.1528	0.0760	0.0040	0.0333
4	0.2081	0.0106	0.0925	0.2878	0.0177	0.1519	0.0762	0.0040	0.0334
5	0.2046	0.0105	0.0908	0.2951	0.0181	0.1552	0.0764	0.0040	0.0334
Mean	0.2060	0.0105	0.0917	0.2907	0.0179	0.1536	0.0765	0.0040	0.0335
Var	$2.3{\cdot}10^{-6}$	$5.7{\cdot}10^{-9}$	$5{\cdot}10^{-7}$	$1{\cdot}10^{-5}$	$4{\cdot}10^{-8}$	$2.1{\cdot}10^{-6}$	$5.9{\cdot}10^{-7}$	$1.4{\cdot}10^{-9}$	$5.4{\cdot}10^{-8}$
LATTICE	0.2090	0.0107	0.0925	0.2929	0.0180	0.1544	0.0829	0.0044	0.0366

Table 9: **LATTICE with TDA text preprocessing:** performance of LATTICE with TDA with text preprocessing in different datasets. The LATTICE baseline is provided in the last line to facilitate ease of comparison.

	Musical instruments			Digital music			Baby		
Iteration	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20
1	0.2056	0.0105	0.0916	0.2965	0.0182	0.1563	0.0757	0.0040	0.0335
2	0.2077	0.0106	0.0936	0.2918	0.0179	0.1533	0.0761	0.0040	0.0335
3	0.2056	0.0105	0.0930	0.2927	0.0180	0.1534	0.0777	0.0041	0.0339
4	0.2063	0.0106	0.0924	0.2965	0.0182	0.1561	0.0747	0.0039	0.0328
5	0.2077	0.0106	0.0920	0.2949	0.0181	0.1560	0.0777	0.0041	0.0340
Mean	0.2066	0.0106	0.0925	0.2945	0.0181	0.1550	0.0764	0.0040	0.0335
Var	$1.1{\cdot}10^{-6}$	$2.8{\cdot}10^{-9}$	$6.4{\cdot}10^{-7}$	$4.6{\cdot}10^{-6}$	$1.7{\cdot}10^{-8}$	$2.3{\cdot}10^{-6}$	$1.7{\cdot}10^{-6}$	$5{\cdot}10^{-9}$	$2.3{\cdot}10^{-7}$
LATTICE	0.2090	0.0107	0.0925	0.2929	0.0180	0.1544	0.0829	0.0044	0.0366

Table 10: **LATTICE with TDA image and text preprocessing:** performance of LATTICE with TDA with image and text preprocessing in different datasets. The LATTICE baseline is provided in the last line to facilitate ease of comparison.

	Musical instruments			Digital music			Baby		
Iteration	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20
1	0.1790	0.0092	0.0807	0.2207	0.0137	0.1101	0.0623	0.0033	0.0274
2	0.1790	0.0092	0.0807	0.2228	0.0138	0.1130	0.0623	0.0033	0.0274
3	0.1790	0.0092	0.0807	0.2234	0.0139	0.1129	0.0623	0.0033	0.0273
4	0.1824	0.0094	0.0802	0.2233	0.0139	0.1128	0.0624	0.0033	0.0274
5	0.1783	0.0092	0.0805	0.2215	0.0138	0.1104	0.0622	0.0033	0.0274
Mean	0.1796	0.0093	0.0805	0.2223	0.0138	0.1118	0.0623	0.0033	0.0274
Var	$2.5{\cdot}10^{-6}$	$4.7{\cdot}10^{-9}$	$4.4{\cdot}10^{-8}$	$1.4{\cdot}10^{-6}$	$4.7{\cdot}10^{-9}$	$2.2{\cdot}10^{-6}$	$6.1{\cdot}10^{-9}$	$1.5{\cdot}10^{-11}$	$6.7{\cdot}10^{-10}$
LATTICE	0.2090	0.0107	0.0925	0.2929	0.0180	0.1544	0.0829	0.0044	0.0366

Table 11: **LATTICE with TDA in the original graph:** performance of LATTICE with TDA on the original graph in different datasets. The LATTICE baseline is provided in the last line to facilitate ease of comparison.

shown in Tabs. 13, 14 and 15 for Musical Instruments, Digital Music and Baby, respectively.

	Musical instruments			Digital music			Baby		
Iteration	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20	R@20	P@20	NDCG@20
1	0.1831	0.0094	0.0803	0.2214	0.0137	0.1103	0.0613	0.0032	0.0271
2	0.1845	0.0095	0.0807	0.2194	0.0136	0.1095	0.0616	0.0033	0.0271
3	0.1831	0.0094	0.0803	0.2219	0.0138	0.1128	0.0614	0.0032	0.0271
4	0.1838	0.0094	0.0806	0.2201	0.0137	0.1125	0.0613	0.0032	0.0271
5	0.1838	0.0094	0.0806	0.2214	0.0138	0.1107	0.0616	0.0033	0.0271
Mean	0.1836	0.0094	0.0805	0.2208	0.0137	0.1111	0.0614	0.0032	0.0271
Var	$3.4{\cdot}10^{-7}$	$8.6{\cdot}10^{-10}$	$2.8{\cdot}10^{-8}$	$1.1{\cdot}10^{-6}$	$5.2{\cdot}10^{-9}$	$2{\cdot}10^{-6}$	$2.3{\cdot}10^{-8}$	$6.1{\cdot}10^{-11}$	$1.3{\cdot}10^{-9}$
LATTICE	0.2090	0.0107	0.0925	0.2929	0.0180	0.1544	0.0829	0.0044	0.0366

Table 12: **LATTICE with TDA on each graph:** performance of LATTICE with TDA on each original graph in different datasets. The LATTICE baseline is provided in the last line to facilitate ease of comparison.

	Musical instruments				
Nodes removed	R@20	P@20	NDCG@20	Time (s)	Epochs
1/5	0.0246	0.0013	0.0092	85	85
1/10	0.0275	0.0014	0.0094	64	65
1/15	0.0241	0.0012	0.0094	65	70
1/20	0.0202	0.0010	0.0073	80	85
1/25	0.0294	0.0015	0.0106	72	75
LATTICE	0.2090	0.0107	0.0925	149	189

Table 13: **LATTICE with node dropping:** performance of LATTICE with node dropping in the Musical Instruments dataset. The LATTICE baseline is provided in the last line to facilitate ease of comparison.

	Digital Music				
Nodes removed	R@20	P@20	NDCG@20	Time (s)	Epochs
1/5	0.0059	0.0004	0.0031	1077	60
1/10	0.0066	0.0004	0.0030	990	65
1/15	0.0061	0.0004	0.0028	1049	65
1/20	0.0056	0.0003	0.0022	1719	115
1/25	0.0066	0.0004	0.0026	1055	65
LATTICE	0.2929	0.0180	0.1543	3413	556

Table 14: **LATTICE with node dropping:** performance of LATTICE with node dropping in the Digital Music dataset. The LATTICE baseline is provided in the last line to facilitate ease of comparison.

	Baby				
Nodes removed	R@20	P@20	NDCG@20	Time (s)	Epochs
1/5	0.0038	0.0002	0.0012	14023	60
1/10	0.0035	0.0002	0.0013	12568	60
1/15	0.0053	0.0003	0.0017	14439	70
1/20	0.0038	0.0002	0.0016	21936	110
1/25	0.0040	0.0002	0.0012	12610	60
LATTICE	0.0825	0.0043	0.0365	4242	118

Table 15: **LATTICE with node dropping:** performance of LATTICE with node dropping in the Baby dataset. The LATTICE baseline is provided in the last line to facilitate ease of comparison.