NTNU – Trondheim
Norwegian University of
Science and Technology

On the Mapper Algorithm

A study of a new topological method for data analysis

Roar Bakken Stovner

Master of Science in Physics and Mathematics
Submission date: June 2012
Supervisor: Nils A. Baas, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

Mapper is an algorithm for describing high-dimensional datasets in terms of simple geometric objects. We give a new definition of Mapper, with which we are able to prove that Mapper is a functor and that Mapper is a homotopy equivalence for certain "nice" input data.

To establish these results we describe the statistical theory of *functorial clustering* and the topological machinery of *homotopy colimits*.

At the end of the document we show, by means of numerical experiments, that the functoriality of Mapper is useful in applications.

Sammendrag

Mapper er en algoritme for å beskrive høyeredimensjonale datamengder som enkle geometriske objekter. Vi presenterer en ny definisjon av Mapper og med denne beviser vi at Mapper er en funktor samt at Mapper er en homotopiekvivalens for visse typer "pene" inndata

For å stadfeste disse resultatene beskriver vi teorien for *funktorielle klyngealgoritmer* og det topologiske maskineriet som *homotopikogrenser* utgjør.

Mot slutten av oppgaven viser vi med numeriske eksperimenter at Mappers funktorialitet er nyttig i anvendelser.

Preface

I write this master's thesis in the spring semester of 2012, under guidance of Professor Nils A. Baas at the Norwegian University of Science and Technology in Trondheim. This work is my contribution to the course "TMA4900 Masteroppgave i matematikk" and should be equivalent to one regular semester of full-time studies.

My work with Nils started in the autumn of 2011 when I under his guidance wrote a summary of the basic theory of persistent homology. This gave me a foothold in the field of statistical topology and I wanted to continue along these lines. The field is new and still ripe with unexplored ideas, so it was hard to choose any one particular topic. I made my decision after mentioning to Nils that I did not want a thesis wholly devoted to applications, I would rather use the applications as an incentive to delve into a theoretical subject. He promptly replied that the "Mapper algorithm" could be regarded as a certain kind of homotopy colimit and that this was an observation about which nothing had so far been written. I was soon hard at work, despite a gentle warning that homotopy colimits is not an easy subject.

In retrospect there is much that could have been done differently. There are many parts of the document which I know can be improved, but which I did not get time to attend to, mostly because I in the last weeks before deadline found out how much work lies behind the writing of proper mathematical exposition. I content myself with mentioning some possible improvements: The last section with numerical experiments leaves me feeling unsatisfied, there are so many ways I could have explored the properties of Mapper which are not included. In the entire document I could have interspersed more examples to illustrate the theory at hand.

This last year I have been in almost continuous company of Truls Bakkejord Ræder, a fellow student and my friend. I thank him for proof-reading this document, for taking on all too many mathematics courses with me, for discussing mathematics with me any time I wanted to, and for occasionally getting me away from mathematics when I needed to.

I thank Nils, my advisor, for the many talks in his office, which have not all been about mathematics, and after these talks I have every time felt inspired to get on with my work. His view on mathematics has greatly shaped mine.

In late January I started working on the thesis and now, in the middle of June, I am finished. I am finished not just with this thesis, but — at least for now — with mathematics at university level. The next years I will be working as a teacher, a teacher of mathematics and science.

I am very grateful for the richness mathematics has added to my life.

Roar Bakken Stovner June 18, 2012

Contents

List of Figures

1. Introduction

Because of the advancements in technology we are collecting information at an ever faster pace. This creates a need for techniques with which we can understand all the information collected. A feature which is common for much of the new information is its high dimensionality. For instance, with new micro-array gene sequencing technology we can extract a fingerprint of the genes of possibly cancerous tissue, each such fingerprint consisting of literally thousands of numbers, making one such fingerprint a point in a vector space of thousands of dimensions [6]. An important question is if we can use this data to decide if the tissue is malignant or benign. This is a statistical problem, but with such high-dimensional data we lose one of the most potent statistical tools we have — good plots. So even though classical statistical methods have proved themselves useful, there are certainly improvements which can be made so that we can better cope with the new challenges that high-dimensional data present.

Topologists at Stanford, lead by Professor Gunnar Carlsson, have the last decade developed exciting new techniques which help us understand high-dimensional data and we point the reader to Carlsson's survey article "Topology and Data" [3] to get an overview of the field.

The term we favor for high-dimensional data sets is **point clouds**, and we define a point cloud to be nothing but a finite metric space. We think of the point cloud as being sampled from a manifold, or even a general topological space, and many questions in statistical topology can be seen as attempts to recreate the underlying manifold from which the point cloud has been sampled. This is also the case with **Mapper**, one of the new algorithms developed during the last decade, and the algorithm which we study in this Master's thesis. The goal of Mapper is to recover a low-dimensional representation of a point cloud, that is, the goal of Mapper is to create a good plot. An example is found on the front page of this document; a plot created by using Mapper on a 2000-dimensional point cloud containing gene samples from prostate cancer tissue.

This document is divided in three parts: Background material in Part I, a discussion of Mapper in Part II, and some numerical experiments in Part III.

In Part I we present some topics which we need in order to understand Mapper. Covers of topological spaces and clustering methods are the main ingredients of Mapper, and these are presented in Chapter 4 and Chapter 5. By defining Mapper in a non-standard way one is led to consider homotopy colimits, which is presented in Chapter 3. We also want to use the term *persistence* from time to time, so we define this in Chapter 2.

In Part II Mapper is presented. Chapter 8 and Chapter 9 are the most important parts of this Master's thesis and contain all of the new work. Most of the other material are included in order to understand these two chapters or to back up the claims made in these two chapters. In Chapter 8 we define Mapper in a new way, making the algorithm a functor almost by definition. In Chapter 9 we prove an analogue to the classic nerve theorem which helps explain why Mapper works.

In Part III we present the our computer implementation of the Mapper algorithm and perform some numerical experiments to show how the algorithm works. We also highlight how the new results in Chapter 8 increase our understanding also in applications.

Part I.

Background

2. Persistence and persistence objects

Persistence is a buzzword in statistical topology and the idea is spreading to many areas of the field from where it first originated, in *persistent homology* [3]. We will not have the opportunity to study persistence thoroughly, but at the end of this document, when summarizing our ideas and tying our work together with the rest of the field, we will use the term. Also, the definition of functorial clustering in Section 5.3 is best stated using the word "persistence". So, even though we will have little use of the terminology and could do without, we feel it is best to use it, if not for any other reason than to comply with the terminology which has become standard in the field.

Let $\mathscr{C}$ be an arbitrary category and let P be a partially ordered set. We consider the poset P as a category in the usual way. A **P-persistence object in $\mathscr{C}$** is a functor $P \to \mathscr{C}$. The collection of P-persistence objects in $\mathscr{C}$ form a category in which morphisms between P-persistence objects are natural transformations between them.

Example 2.0.1. If we choose P to be a totally ordered set, for instance $\mathbb{N}$, then a P-persistence object in $\mathscr{C}$ would be a linear diagram. In the case of $P = \mathbb{N}$ we could display the persistence object as

$$\cdots \longrightarrow C_{n-1} \xrightarrow{f_{n-1}} C_n \xrightarrow{f_n} C_{n+1} \longrightarrow \cdots,$$

where the C_i and f_i are objects and morphisms in $\mathscr{C}$ respectively. We have omitted drawing all the compositions of functions, although they are part of the persistence object. □

So P-persistence objects in $\mathscr{C}$ are simply diagrams in $\mathscr{C}$ which are shaped like the poset P.

The idea of persistence is often used when we have a problem of choosing proper parameter values. Say that we have a method F for producing a

geometric representation of some point cloud. The method F can be, for instance, persistent homology or Mapper. These methods are initialized by some parameter $p \in \mathbb{R}$ and it is important to study how sensitive F is to changes in p. By establishing that $F(-)$ is an $\mathbb{R}$-persistence object we have for $p_0 \leq p_1$ an induced map $F(p_0) \to F(p_1)$ and the existence of such a map is treated as a kind of stability of F, especially if we can prove some sort of well-behavedness of the induced map. If some geometric feature of $F(p_0)$, for example nontrivial homology, is also present in $F(p_1)$ we say that the feature **persists** from p_0 to p_1. Features which persists through many parameter choices are considered "true" features of the point cloud, while features which quickly appear and disappear are considered random fluctuations or artifacts of the method F.

3. Homotopy colimits

We can define the Mapper algorithm of Part II in an abstract way as a homotopy colimit, and this is beneficial because it makes many of Mapper's properties more transparent. Therefore, we will in this chapter give the definition of the homotopy colimit and derive some of its most basic properties. Most introductions to the homotopy colimit uses the language of model categories, which is an interesting subject in itself, but which we shall have no further use of, so we choose to rather give the definition of the homotopy colimit in as direct a fashion as possible, and in a way which is easy to compute.

A problem we had when learning about homotopy colimits was that in every computation of a homotopy colimit some details seemed to be left out, except maybe in the simplest possible cases. We have therefore gone to great lengths to write out every minute detail of at least some of our calculations, accepting that the notation becomes cumbersome and that one stand in danger of losing sight of the basic idea behind homotopy colimits.

Our exposition is based on Dugger [7].

Introduction

We need homotopy colimits because the normal colimits do not behave well under homotopy equivalences. To see this, consider the following two diagrams of spaces

$$\mathbb{D}^2 \longleftarrow \mathbb{S}^1 \longrightarrow \mathbb{D}^2$$

$$* \longleftarrow \mathbb{S}^1 \longrightarrow *$$

where the asterisk $*$ denote a one-point space. The arrows in the upper diagram are functions including $\mathbb{S}^1$ onto the boundary of $\mathbb{D}^2$. Since the spaces occupying the same places in the above two diagrams are homotopy equivalent ($\mathbb{D}^2 \simeq *$ and $\mathbb{S}^1 \simeq \mathbb{S}^1$), one might think that the colimits of the

diagrams are homotopy equivalent as well. This is not the case since the colimit of the upper diagram is equivalent to $\mathbb{S}^2$ and the colimit of the lower diagram is equivalent to $*$. Wee see that in the colimit of the lower diagram any trace of $\mathbb{S}^1$ has been lost — $\mathbb{S}^1$ has simply been squashed down to a point! This illustrates what goes wrong with the normal colimit, it squashes spaces together a little too much.

This example points out the need for a homotopy-friendly colimit; a colimit such that component-wise homotopy equivalent diagrams have homotopy equivalent colimits. In order to achieve this we introduce the idea of "gluing along homotopies" or "gluing along paths". The normal colimit of the diagram

$$A \xleftarrow{f} X \xrightarrow{g} B$$

is constructed by taking $A \coprod B$ and identifying the points $f(x)$ and $g(x)$ of for all x in X, whereas the homotopy colimit is constructed by taking $A \coprod B \coprod (X \times [0,1])$ and identifying $f(x)$ with $(x,0)$ and $g(x)$ with $(x,1)$ for all $x \in X$. You can think of this as attaching $f(x)$ to $g(x)$, but with the cylinder $X \times [0,1]$ stuck in the middle, that is, $f(x)$ and $g(x)$ are glued together along a path.

So in essence the idea is still to glue points to each other, but to allow enough leeway for homotopy deformations to take place, and this disallows the squashing that occurred in the above example. This leeway was given by not gluing two spaces directly together, but rather onto each end of a unit interval. In general, though, we might have to glue more than two spaces together and then the unit interval does not suffice as intermediary because it has got only two ends onto which we can glue. The solution is to use the standard simplices Δ^n as intermediaries; because Δ^n has a boundary with $[n+1]$ parts we can glue $n+1$ spaces onto Δ^n.

Unfortunately, keeping track of how all the spaces of our diagram are supposed to be glued together is a combinatorial challenge. Therefore, we first associate a combinatorial device with our diagram $\mathscr{D}$, its **simplicial replacement**, a special case of a **simplicial object**, and then define the homotopy colimit of $\mathscr{D}$ to be the **geometric realization** of the simplicial replacement of $\mathscr{D}$. All these notions are discussed in the following few sections.

It is stated directly above how we plan to construct the homotopy colimit,

namely as the geometric realization of the simplicial replacement of a diagram. This construction, however, is only a *choice* of homotopy colimit and we could have made other choices. What we mean by this is that the defining property of homotopy colimits, that component-wise homotopy equivalent diagrams have homotopy equivalent homotopy colimits, does not define the homotopy colimit up to homeomorphism type, only up to homotopy type.

Note. For us, the word "homotopy colimit" can be misleading, as we shall have no opportunity to see the universal property which the homotopy colimit fulfills. For our purposes, it suffices to think of the homotopy colimit as a space-level realization of a diagram, enabling us to study properties of the diagram with homotopical methods. The same sentiment has been expressed by Hatcher [10, Chapter 4.G]:

> [...] This has given rise to the rather unfortunate name of 'hocolim' for ΔX, short for 'homotopy colimit'. In preference to this we have chose the term 'realization', both for its intrinsic merits and because ΔX is closely related to what is called the geometric realization of a simplicial space. ...

Unlike Hatcher, we shall use the term homotopy colimit. □

3.1. Simplicial objects and the simplex category

The proper category theoretic analogue of ordered simplices is found in the **simplex category** Δ (also called the **cosimplicial indexing category**).

The category Δ is defined to be the category of totally ordered finite sets with (weakly) order preserving maps. Note that all totally ordered finite sets of cardinality n are isomorphic under a unique isomorphism, so we lose no generality by choosing to work with the skeletal subcategory of Δ consisting of the objects

$$[n] = \{0, 1, \dots, n-1, n\},$$

where the ordering is given in the natural way. We will do this tacitly from now on. Also, it is not hard to convince oneself that that all morphisms of Δ can be expressed as the finite composition of two "archetype morphisms", the **coface maps** $d^i_n : [n-1] \to [n]$ and the **codegeneracy maps** $s^i_n : [n+1] \to [n]$. The map d^i_n "skips" the element i and the map s^i_n "repeats" the element

i, that is,

$$d_n^i(k) = \begin{cases} k & \text{if } k < i \\ k+1 & \text{if } k \geq i \end{cases} \qquad s_{n+1}^i(k) = \begin{cases} k & \text{if } k < i \\ i & \text{if } k = i \text{ or } k = i+1 \\ k-1 & \text{if } k > i+1 \end{cases}.$$

It is customary to drop the lower index and just write d^i and s^i.

We often think of the simplex category Δ by its image under the covariant inclusion of categories $\Delta \hookrightarrow Top$ which maps $[n]$ to the n-dimensional standard simplex Δ^n and which maps morphisms $\varphi : [n] \to [k]$ to the induced linear map $\Delta^n \to \Delta^k$ which coincide with φ on the vertices of Δ^n.

The next definition is used to impose the simplicial structure of the simplex category Δ on other categories $\mathscr{C}$. A **simplicial object in** $\mathscr{C}$ is a *contra*variant functor $\Delta \to \mathscr{C}$ and a **simplicial morphism in** $\mathscr{C}$ is a natural transformation between two simplicial objects in $\mathscr{C}$. These objects and morphisms together define a category. Simplicial objects in *Top* are called **simplicial spaces** and simplicial objects in Set are called **simplicial sets** and the corresponding categories are denoted *sSpace* and *sSet*.

Note. We often write $X_\bullet$ for a simplicial object and take X_i to mean $X_\bullet([i])$, and take d_i and s_i to mean $X_\bullet(d^i)$ and $X_\bullet(s^i)$. The indexes of the morphisms change place in order to remind us of the contravariance (exactly as for homology and *co*homology). □

Simplicial sets are abstractions of simplicial complexes. In order to see this, think of a simplicial set $X_\bullet$ as a collection of n-simplices, one for each element x of X_n, with the i^{th} face of x being given by the face map d_i^n. The degeneracy maps s_i^n are a little more artifical sounding than the face maps; the i^{th} degeneracy of x, $s_i^n x$, correspond to an $n+1$-simplex which we think of as being equal to x, but with an added degenerate i^{th} dimension. This abstraction is formalized (and generalized to simplicial spaces) in Section 3.2.

3.2. Geometric realization

Let $X_\bullet$ be a simplicial space. The geometric realization of a simplicial space, which we define shortly, unifies the spaces X_i with their simplicial structure into a topological space $|X_\bullet|$. In the space $|X_\bullet|$ the spaces X_i

are glued together along standard simplices in a way dictated by the face maps $X_i \to X_{i-1}$. There is also an action of the geometric realization on *sSpace*-morphisms turning $|-|$ into a covariant functor. The first to give this definition the geometric realization was Milnor [13] and we follow his notation.

Let $X_\bullet$ be a simplicial space and Δ^i the i-dimensional standard simplex. The **geometric realization $|X_\bullet|$ of $X_\bullet$** is the space

$$\coprod_{i=0}^{\infty} X_i \times \Delta^i$$

subject to the following identifications

$$(d_i x, t) \sim (x, d^i t)$$
$$(s_i x, t) \sim (x, s^i t).$$

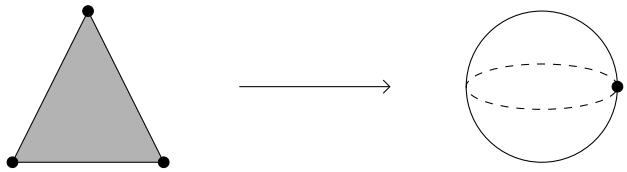
We will denote the equivalence class which (x,t) belongs to by $|x,t|$.

Note. The above equivalence relation is the same as the one generated by identifying $(\varphi x, t) \sim (x, \varphi t)$ for all maps φ of Δ, that is, for all order preserving maps. That these equivalence relations are equal is an easy consequence of the fact that any order preserving map φ is a composition of a number of d_i and s_i. □

Note. This definition extends to simplicial sets $X_\bullet$ by equipping the sets X_i with the trivial topology. □

We think of the identifications $(d_i x, t) \sim \left(x, d^i t\right)$ as "gluing along faces" and of the identifications $(s_i x, t) \sim \left(x, s^i t\right)$ as "contracting degeneracies". The logic of these names hopefully becomes clear with the next example.

Example 3.2.1. To gain a feel for how the geometric realization works, let us try to construct a simplicial set $X_\bullet$ with geometric realization of the same homotopy type as $\mathbb{S}^2$. The idea we will use is to simulate the way $\mathbb{S}^2$ is most often realized as a CW-complex, that is, as a 2-cell whose boundary is collapsed to a point.



We need a 0-simplex $\alpha \in X_0$ representing the base point and a 2-simplex $\beta \in X_2$ representing the two-cell. We want no 1-simplex in our geometric realization, so the set X_1 must consist of only one element $X_1 = \{\alpha\}$ because then this element must be a degeneracy of $\alpha \in X_0$ and is hence collapsed to a point. The identification $(s_i\alpha, \Delta^1) \sim (\alpha, s^i\Delta^1)$ takes care of this. Also, there should exist a $\beta \in X_2$ which is not a degeneracy of $\alpha \in X_1$ because then $\{\beta\} \times \Delta^2$ would be collapsed to a point as well. The simplicial set $X_\bullet$ therefore consists of the following sets

$$X_0 = X_1 = \{\alpha\} \text{ and } X_i = \{\alpha, \beta\} \text{ for } i \geq 2,$$

and all the face and degeneracy morphisms map α to α and β to β except d_0^2, d_1^2 and d_2^2 which map β to α. These three maps identify the three faces of $\{\beta\} \times \Delta^2$ with $\{\alpha\} \times \Delta^1$ and, as we remember, $\{\alpha\} \times \Delta^1$ is identified with a point.

Therefore, the only non-degenerate parts of $|X_\bullet|$ are $\{\alpha\} \times \Delta^0$ and $\{\beta\} \times \Delta^2$ and the faces of $\{\beta\} \times \Delta^2$ are identified with $\{\alpha\} \times \Delta^0$. This space is certainly homotopy equivalent to $\mathbb{S}^2$. □

A map of simplicial spaces $f : X_\bullet \to Y_\bullet$ induces a map $|f| : |X_\bullet| \to |Y_\bullet|$ given by $|x, t| \to |f(x), t|$. This assignment of maps respects identities and composition, so we have a covariant functor $|-|$: *sSpace* $\to$ *Top*.

3.3. Simplicial replacement

In the introduction to this chapter we noted that in order to define the homotopy colimit we need a combinatorial scheme which specifies how spaces are supposed to be glued onto the boundary of standard simplices. This scheme is the *simplicial replacement* of the diagram. Intuitively, the 0-simplices of the simplicial replacement are the spaces of the diagram and the n-simplices are all n-fold compositions of morphisms of the diagram. The n^{th} face map "deletes" a morphism, creating an $n-1$-fold composition, and the n^{th} degeneracy map inserts an identity morphism, creating an $n+1$-simplex.

Note. The simplicial replacement is related to, and is indeed very similar to, the category theoretic **nerve**, but whereas the nerve constructs a simplicial *set* from a diagram in *any small category* the simplicial replacement constructs a simplicial *space* from a diagram in *Top*. The simplicial replacement

and the nerve coincide for diagrams of one-point spaces and we will take this as the definition of the nerve of a diagram. □

Let $\mathscr{C}$ be a small category with objects c_i. The **simplicial replacement** (or **simplicial bar construction**) of a diagram $D : \mathscr{C} \to Top$ is denoted $srep\,D$ and is the simplicial space which acts on any object $[n]$ in Δ by

$$srep\,D([n]) = \coprod_{c_0 \to c_1 \to \ldots \to c_n} D(c_0). \tag{3.1}$$

By this we mean a disjoint union which ranges over any n-fold composition of $\mathscr{C}$-morphisms which starts in c_0 (for all choices of $c_0 \in Ob(\mathscr{C})$). To define the face and degeneracy maps of $srep\,D$ we need some notation. We refer to the $D(c_0)$ indexed by

$$\sigma = \left(c_0 \xrightarrow{f_0} c_1 \xrightarrow{f_1} \ldots \to c_{n-1} \xrightarrow{f_{n-1}} c_n \right)$$

as $D(c_0)_\sigma$ and define "deletions" $d_i(\sigma)$ and "repetitions" $s_i(\sigma)$ as

$$d_i(\sigma) = \begin{cases} c_1 \xrightarrow{f_1} c_2 \to \ldots \to c_n & \text{for } i = 0 \\ c_0 \to \ldots \to c_{i-1} \xrightarrow{f_{i-1} \circ f_i} c_{i+1} \to \ldots \to c_n & \text{for } 0 < i < n \\ c_0 \to \ldots \to c_{n-2} \to c_{n-1} & \text{for } i = n \end{cases}$$

$$s_i(\sigma) = \begin{cases} c_0 \xrightarrow{id} c_0 \xrightarrow{f_0} c_1 \to \ldots \to c_n & \text{for } i = 0 \\ c_0 \to \ldots \to c_{i-1} \xrightarrow{f_{i-1}} c_i \xrightarrow{id} c_i \xrightarrow{f_i} c_{i+1} \to \ldots \to c_n & \text{for } 0 < i < n \\ c_0 \to c_1 \to \ldots \to c_n \xrightarrow{id} c_n & \text{for } i = n \end{cases}.$$

We also write d_i^* for $srep\,D(d_i)$ and s_i^* for $srep\,D(s_i)$. To define a map on a disjoint union it is enough to define it on each of its components and therefore the morphisms d_i^* and s_i^* can be specified by

$$d_i^*\Big|_{D(c_0)_\sigma} = \begin{cases} D(c_0)_\sigma \xrightarrow{id} D(c_0)_{d_i(\sigma)} & \text{for i > 0} \\ D(c_1)_\sigma \xrightarrow{f_0} D(c_1)_{d_i(\sigma)} & \text{for i = 0} \end{cases} \qquad s_i^*\Big|_{D(c_0)_\sigma} = D(c_0)_\sigma \xrightarrow{id} D(c_0)_{s_i(\sigma)}.$$

That this action on the maps of Δ makes $srep\,D$ a simplicial space can be verified easily.

The definition of the simplicial replacement is admittedly hard to come to grips with, so in order to help the reader we present an example worked out in full.

Example 3.3.1. In this example we write out the simplicial replacement of a diagram $D : \mathscr{P} \to Top$ where $\mathscr{P}$ is the pre-pushout category,

$$\mathscr{P} = \left(1 \xleftarrow{i} 2 \xrightarrow{j} 3\right).$$

We write D as $\left(A \xleftarrow{f} X \xrightarrow{g} B\right)$ with $A = D(1)$, $f = D(i)$ and so on. We have chosen to not write the identity morphisms for typographical reasons. We list the "0-simplices"

$$srep\, D([0]) = A \coprod X \coprod B,$$

and the "1-simplices" (with the convention that $X_{(X \to A)}$ means "the copy of X indexed by the 1-fold composition $X \to A$")

$$srep\, D([1]) = X_{\left(X \xrightarrow{f} A\right)} \coprod X_{\left(X \xrightarrow{g} B\right)} \coprod A_{\left(A \xrightarrow{id} A\right)} \coprod B_{\left(B \xrightarrow{id} B\right)} \coprod X_{\left(X \xrightarrow{id} X\right)}.$$

and the "2-simplices"

$$\begin{aligned} srep\, D([2]) = \quad & X_{\left(X \xrightarrow{id} X \xrightarrow{f} A\right)} \coprod X_{\left(X \xrightarrow{f} A \xrightarrow{id} A\right)} \coprod X_{\left(X \xrightarrow{id} X \xrightarrow{g} B\right)} \coprod X_{\left(X \xrightarrow{g} B \xrightarrow{id} B\right)} \\ & \coprod X_{\left(A \xrightarrow{id} A \xrightarrow{id} A\right)} \coprod X_{\left(X \xrightarrow{id} X \xrightarrow{id} X\right)} \coprod X_{\left(B \xrightarrow{id} B \xrightarrow{id} B\right)}. \end{aligned}$$

The higher order simplices are constructed in a similar fashion with the proper amount of identity morphisms added at either the beginning or the end.

We now turn to the definition of the morphisms $srep\, D(d^i_n)$, which we denote with $d^{n\,*}_i$. We only write out the morphisms restricted to one component

of the coproduct as this is enough to specify them completely.

$$d_1^{1*} = \left(id : X_{\left(X \xrightarrow{f} A\right)} \to X \right)$$

$$d_0^{1*} = \left(f : X_{\left(X \xrightarrow{f} A\right)} \to A \right)$$

$$d_2^{2*} = \left(id : X_{\left(X \xrightarrow{f} A \xrightarrow{id} A\right)} \to X_{\left(X \xrightarrow{f} A\right)} \right)$$

$$d_1^{2*} = \left(id : X_{\left(X \xrightarrow{f} A \xrightarrow{id} A\right)} \to X_{\left(X \xrightarrow{f \circ id} A\right)} \right) = \left(id : X_{\left(X \xrightarrow{f} A \xrightarrow{id} A\right)} \to X_{\left(X \xrightarrow{f} A\right)} \right)$$

$$d_0^{2*} = \left(id : X_{\left(X \xrightarrow{f} A \to A\right)} \xrightarrow{id} A_{\left(A \xrightarrow{id} A\right)} \right).$$

The continuous functions s_i^{n*} are always identity morphisms and their codomains are always found by composing with an identity morphism (in the subscript). This makes the s_i^{n*} a little easier to write down, so we choose to only present two of them:

$$s_1^{1*} = \left(X_{\left(X \xrightarrow{f} A\right)} \xrightarrow{id} X_{\left(X \xrightarrow{f} A \xrightarrow{id} A\right)} \right)$$

$$s_0^{1*} = \left(X_{\left(X \xrightarrow{f} A\right)} \xrightarrow{id} X_{\left(X \xrightarrow{id} X \xrightarrow{f} A\right)} \right).$$

□

The simplicial replacement is indeed a functor. Let $C : \mathscr{C} \to Top$ and $D : \mathscr{C} \to Top$ be diagrams and let $\varphi : C \to D$ be a map of diagrams. There is an induced map $srep\left(\varphi\right) : srep\left(C\right) \to srep\left(D\right)$ given by sending an n-fold composition of morphisms $f_n \circ \ldots \circ f_1$ of C to the n-fold composition of morphisms in D given by $\varphi(f_n \circ \ldots \circ f_1)$. It is easy to check that this assignment commutes with face and degeneracy maps, such that $srep\left(\varphi\right)$ is well-defined map of simplicial spaces, and also that the assignment $srep(-)$ preserves both composition and identity morphisms.

In summary, we have that $srep : [\mathscr{C}, Top] \to sSpace$ is a covariant functor. (The symbols $[\mathscr{C}, Top]$ means the category of functors from $\mathscr{C}$ to Top.)

3.4. Homotopy colimit

Let $\mathscr{C}$ be any small category.

The **homotopy colimit** is the covariant functor $\operatorname{hocolim}\colon [\mathscr{C}, Top] \to Top$ defined as the composition of the geometric realization and the simplicial replacement,

$$\operatorname{hocolim}(-) = |srep\,(-)|.$$

We state the following important property of homotopy colimits without proof. A proof written for this particular construction of the homotopy colimit can be found in Dugger [7, Proposition 4.7].

Proposition 3.4.1 (The fundamental property of homotopy colimits)**.** *Let $X_\bullet\colon \mathscr{C} \to Top$ and $Y_\bullet\colon \mathscr{C} \to Top$ be diagrams and let $f\colon X \to Y$ be a natural transformation. If all components of f are homotopy equivalences, then the induced map*

$$\operatorname{hocolim}(f)\colon \operatorname{hocolim}(X_\bullet) \to \operatorname{hocolim}(Y_\bullet)$$

is a homotopy equivalence. □

The following example contains a detailed computation of a homotopy colimit.

Example 3.4.1. We continue Example 3.3.1 where we wrote the simplicial replacement of the following diagram D of spaces,

$$D = \left(A \xleftarrow{f} X \xrightarrow{g} B\right).$$

The goal of this example is to write down the space $\operatorname{hocolim} D$, that is, we want to realize the simplicial space $srep\,D$ geometrically.

Recall that we in Example 3.2.1 argued that the codomains of degeneracy maps can be ignored when taking the geometric realization. These are identified with their domain anyway, and add nothing new to the space $\operatorname{hocolim} D$. This saves us from considering, say, $X_{(X\to X\to A)}$, because it is the codomain of $X_{(X\to A)}$ under a degeneracy map. We say that the image of degeneracy maps are **degeneracies**.

After pruning away all degenerate direct summands of Example 3.3.1, we are left with the following non-degenerate direct summands in $\operatorname{hocolim} D$

$$X \coprod A \coprod B \coprod X_{(X\to A)} \times \Delta^1 \coprod X_{(X\to B)} \times \Delta^1. \tag{3.2}$$

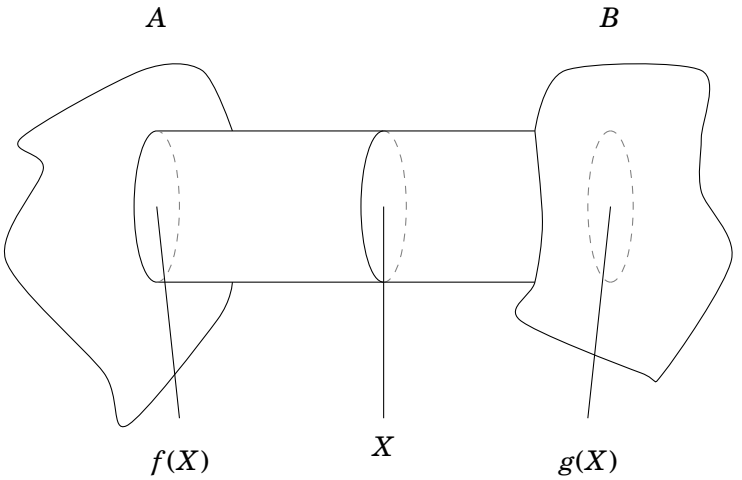
and these are glued together according to face relations. The face relations for $X_{(X\to A)}$ are

$$d_1\left(X_{(X\to A)}\right)\times\Delta^0 \sim X\times d^1\left(\Delta^0\right)$$
$$d_0\left(X_{(X\to A)}\right)\times\Delta^0 \sim X\times d^0\left(\Delta^0\right)$$

which we evaluate to obtain

$$X\times\Delta^0 \sim X_{(X\to A)}\times\{1\}$$
$$f(X)\times\Delta^0 \sim X_{(X\to A)}\times\{0\}.$$

This is an X-shaped cylinder with one end glued onto A. The face relations for $X_{(X\to B)}$ provide a similar cylinder with one end glued onto B. We see that the other ends of these two cylinders are identified with the very same direct summand X of (3.2). The following conceptual picture shows the construction, which the reader might recognize as the **homotopy pushout** of $A\leftarrow X\to B$.



□

Example 3.4.2. With the above calculation of the homotopy pushout, it is easy to recognize the mapping cone of a continuous function $f: X\to Y$ as

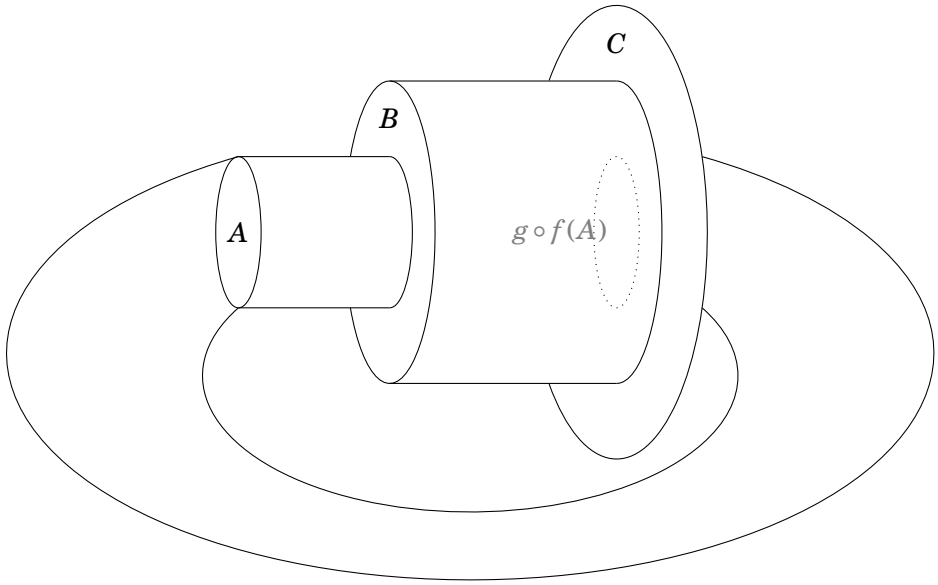


Figure 3.1.: The homotopy colimit of the diagram $A \to B \to C$ consisting of $f : A \to B$ and $g : B \to C$.

the homotopy pushout of the diagram

$$Y \xleftarrow{f} X \to pt.$$

□

The scheme for calculating the homotopy colimit is clear

1. write out all the disjoint unions of the simplicial replacement like in (3.1), but ignore the codomain of degeneracy maps
2. write down identifications given by the face maps.

This is a recipe which we shall have the opportunity to reuse many times.

Example 3.4.3. We now find the homotopy colimit of the diagram

$$A \xleftarrow{f} B \xrightarrow{g} C.$$

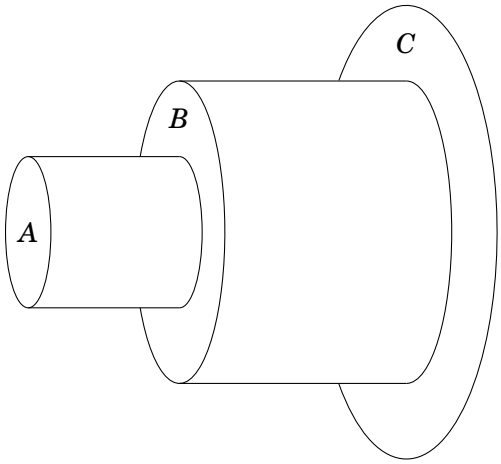


Figure 3.2.: The mapping telescope of $A \to B \to C$.

The purpose of this example is to show how a non-degenerate 2-simplex, like $A_{(A\to B\to C)}$, is treated by the homotopy colimit.

Taking the lead from Example 3.4.1 we know that each of the non-degenerate 1-simplices

$$A_{(A\to B)} \times \Delta[1],\ A_{(B\to C)} \times \Delta[1] \text{ and } A_{\left(A \xrightarrow{g\circ f} C\right)} \times \Delta[1], \tag{3.3}$$

contribute a cylinder to the homotopy colimit, and that they are attached like in Figure 3.1.

The homotopy is given by homotopy deforming the cylinder $A_{A\to C} \times \Delta^1$ along $A_{A\to B\to C} \times \Delta^2$ and leaving the rest of the space W in place. We call this space W. To complete the homotopy colimit we must attach the space $A_{A\to B\to C} \times \Delta^2$ to W as specified by the 2-face maps. This results in the sides of $A_{A\to B\to C} \times \Delta^2$ being glued onto the cylinders $A_{A\to C} \times \Delta^1$, $A_{A\to B} \times \Delta^1$ and $f(A) \times \Delta^1$ respectively. This is the space $\operatorname{hocolim}(A \to B \to C)$ and it is homotopy equivalent to the regular mapping telescope of $A \to B \to C$, pictured in Figure 3.2. □

Example 3.4.4. We saw in Example 3.4.3 that we got a truncated mapping telescope from the sequence of maps $A \to B \to C$. In the same way we get

the regular mapping telescope by taking the homotopy colimit of the infinite chain

$$\ldots \to A_{i-1} \to A_i \to A_{i+1} \to \ldots .$$

This space looks like an infinite extension of the space shown in Figure 3.2. □

Our most important example will be a homotopy colimit taken of an open cover of a topological space, but we will have to wait until the end of the next chapter before studying it, because we need to discuss open covers first.

4. Covers of topological spaces

We define a category *TopCov* of topological spaces with associated coverings and then see how some common constructions using open covers are in fact functors. Covers of topological spaces is the main ingredient of the Mapper algorithm presented in Part II, so these preliminary remarks will be of great importance there. In particular, we want to formulate how the Mapper algorithm is a functor so we construct *TopCov* to act as input category for the Mapper-functor.

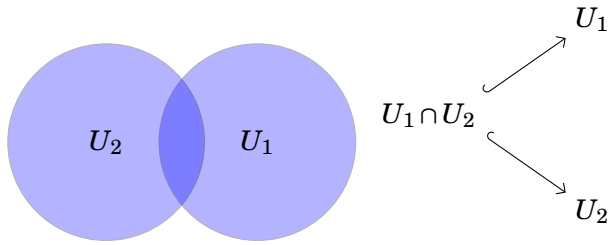
4.1. An open cover as a diagram of spaces

Let $\mathscr{U} = \{U_a\}_{a \in A}$ be an open cover of the topological space X. For any subset σ of A we define U_σ to be the intersection $\bigcap_{a \in \sigma} U_a$. We define the category $X_{\mathscr{U}}$ associated to $\mathscr{U}$ whose objects are the non-empty U_σ for finite subsets σ of A, and whose morphisms are the inclusions $U_\sigma \hookrightarrow U_{\sigma'}$ for subsets σ and σ' of A such that $\sigma' \subseteq \sigma$. This turns $X_{\mathscr{U}}$ into a category, because composition of inclusions are inclusions and identity morphisms are inclusions. We will often regard $X_{\mathscr{U}}$ as a diagram of topological spaces, making $X_{\mathscr{U}}$ an object of the functor category Top^{Cat}.

An example of this construction is shown in Figure 4.1.

4.2. A category of covered topological spaces, *TopCov*

We construct a category of openly covered spaces. The purpose of defining this category is to have a device on which the assignment $\mathscr{U} \to X_{\mathscr{U}}$ is a functor. To have this functor established is important because it ultimately leads to functoriality of the Mapper algorithm, and, as we will argue for later, this means that Mapper is stable under certain changes of input data.



(a) The cover $\mathcal{U}$ of X (b) The diagram $X_{\mathcal{U}}$

Figure 4.1.: Example of the construction $X_{\mathcal{U}}$ for the space $X = U_1 \cup U_2$ and cover $\mathcal{U} = \{U_1, U_2\}$.

Let X and Y be topological spaces and let $\mathcal{U}$ and $\mathcal{V}$ be open covers of X and Y respectively. A **map of covers** $f : (X, \mathcal{U}) \to (Y, \mathcal{V})$ is a continuous function $f : X \to Y$ such that for all $U \in \mathcal{U}$ there exists a $V \in \mathcal{V}$ such that $f(U)$ is contained in V. Composition of maps of covers $f : (X, \mathcal{U}) \to (Y, \mathcal{V})$ and $g : (Y, \mathcal{V}) \to (Z, \mathcal{W})$ is defined by $g \circ f : (X, \mathcal{U}) \to \{Z, \mathcal{W}\}$ where $g \circ f$ is the normal composition of functions. We see that the composition $g \circ f$ is a map of covers. The identity morphism $\mathrm{id}_{(X,\mathcal{U})}$ becomes the identity function on X. Thus, we have a category of topological spaces and coverings and we denote this category by *TopCov*.

Note. Our goal was to create a category on which the $X_{\mathcal{U}}$-construction was a functor and we see in the next section that we with *TopCov* have accomplished our goal. However, there are also other natural choices of categories on which $X_{\mathcal{U}}$ is a functor. For instance, we could define the category *TopCov*$'$, which has the same objects as *TopCov*, but where the morphisms $f : (X, \mathcal{U}) \to (Y, \mathcal{V})$ are those continuous maps $f : X \to Y$ such that for all $V \in \mathcal{V}$ we have $f^{-1}(V) \in \mathcal{U}$. Also on this category is $X_{\mathcal{U}}$ a functor, so we need to argue why we did not choose *TopCov*$'$ over *TopCov*. We therefore present a use-case where a morphism which is not found in *TopCov*$'$ naturally appears.

With the Mapper algorithm we want to study a cover $\mathcal{U}$ of a point cloud X. The input of Mapper is the cover $\mathcal{U}$ of X, and if the cover $\mathcal{U}$ is too granular, Mapper gives a too disconnected representation of X. We have ways of obtaining *coarser* covers $\mathcal{U}'$ of X, that is, covers $\mathcal{U}'$ such that $\mathcal{U}$

refines $\mathscr{U}'$. For such covers, if $\iota\colon (X,\mathscr{U}) \to \left(X,\mathscr{U}'\right)$ is the identity on the underlying space X, then ι is a morphism of *TopCov*, but not of *TopCov*$'$. Therefore, with *TopCov* as input category, we could talk of an induced map $Mapper(\iota) : Mapper(\mathscr{U}) \to Mapper\left(\mathscr{U}'\right)$, but with *TopCov*$'$ not.

This argument is not the final say in the matter — there might be another better suited category *TopCov*$''$ — but for our purposes *TopCov* suffices. □

Functoriality of $X_{\mathscr{U}}$

We now show how to make the assignment $(X,\mathscr{U}) \to X_{\mathscr{U}}$ into a functor from *TopCov* to Top^{Cat}.

Let $f\colon (X,\mathscr{U}) \to (Y,\mathscr{V})$ be a map of covers for the covers $\mathscr{U} = \{U_a\}_{a\in A}$ and $\mathscr{V} = \{V_b\}_{b\in B}$ of X and Y respectively. For a subset σ of A such that U_σ is non-empty we denote by $\widehat{f}(\sigma)$ the set of all $b \in B$ such that $f(U_\sigma) \subseteq V_b$. We have the inclusion $f(U_\sigma) \subseteq V_{\widehat{f}(\sigma)}$. The map of covers f induces a natural transformation between the diagrams $X_{\mathscr{U}}$ and $Y_{\mathscr{V}}$ which takes the node U_σ of $X_{\mathscr{U}}$ to the node $V_{\widehat{f}(\sigma)}$ of $Y_{\mathscr{V}}$ and takes the inclusion $U_\sigma \hookrightarrow U_{\sigma'}$ to the inclusion $V_{\widehat{f}(\sigma)} \hookrightarrow V_{\widehat{f}(\sigma')}$. It is easily checked that this is indeed a natural transformation.

It is easily checked that identity morphisms in *TopCov* induce identity morphisms of Top^{Cat} and that compositions is preserved. Thus, we have a covariant functor $\mathfrak{D}\colon TopCov \to Top^{Cat}$.

Continuous maps and *TopCov*

The category *TopCov* respects continuous functions in the following way: Let $f\colon X \to Y$ be a map and let $\mathscr{V} = \{V_b\}_{b\in B}$ be a cover of Y. The map f and the open cover $\mathscr{V}$ define a cover $f^{-1}(\mathscr{V}) = \{f^{-1}(V_b)\}_{b\in B}$ of X. This cover is open since f is continuous and we say that $f^{-1}(\mathscr{V})$ is the **open cover induced by** f. We now get a map of covers $f\colon \left(X, f^{-1}(\mathscr{V})\right) \to (Y,\mathscr{V})$, which we say is the **map of covers induced by** $f : X \to Y$ **and** $\mathscr{V}$.

4.3. Nerve of a cover

The **nerve of a cover** $\mathscr{U} = \{U_a\}_{a\in A}$ is the abstract simplicial complex $N(\mathscr{U})$ with vertices A where a subset $\{a_0, a_1, \ldots, a_k\}$ of A span a k-simplex if and only if $\bigcap_{i=0}^{k} U_{a_i}$ is non-empty. We regard $N(\mathscr{U})$ as a simplicial set

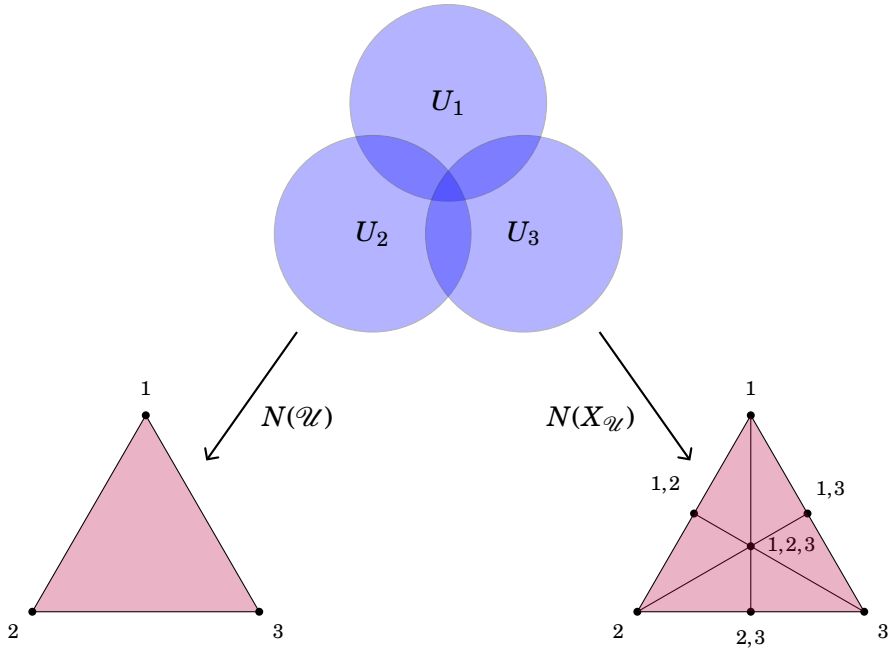


Figure 4.2.: A cover $\mathcal{U} = \{U_1, U_2, U_3\}$ and the realization of the two simplicial sets $N(\mathcal{U})$ and $N(X_{\mathcal{U}})$. The vertex labeled i corresponds to U_i, vertex labeled i,j corresponds to $U_i \cap U_j$ and the vertex i,j,k corresponds to $U_i \cap U_j \cap U_k$. We see that $N(X_{\mathcal{U}})$ is the barycentric subdivision of $N(\mathcal{U})$.

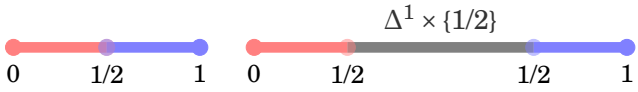
with face maps $d_i(\{a_0, a_1, \ldots, a_k\}) = \{a_0, \ldots, \widehat{a_i}, \ldots, a_k\}$ and degeneracy maps $s_i(\{a_0, \ldots, a_i, \ldots, a_k\}) = \{a_0, \ldots, a_i, a_i, \ldots, a_k\}$. Note that we need to specify a total order on each simplex $\sigma \in A$ for these maps to be well-defined.

The nerve N is a functor $\mathit{TopCov} \to \mathit{sSet}$; that is, there is a map

$$N(f) : N(\mathcal{U}) \to N(\mathcal{V})$$

given by the simplicial map induced by $\widehat{f}$. This map sends the simplex of $N(\mathcal{U})$ spanned by $S \subseteq A$ to the simplex of $N(\mathcal{V})$ spanned by $\widehat{f}(S) \subseteq B$. It is easily checked that $N(f)$ is a map of simplicial sets, that is, it commutes with face and degeneracy maps.

In Section 3.3 we defined a functor N on diagrams, also called a *nerve*. We will now see that the nerve of the cover $\mathcal{U}$ is closely related to the nerve



(a) The unit interval (b) Blowup of the interval

Figure 4.3.: The unit interval covered by $[0,1/2]$ (in red) and $[1/2,1]$ (in blue). In (b) the Mayer-Vietoris blowup of $\mathscr{U}$ is depicted. The intersection of the two covering sets have been "blown up" by a 1-simplex.

of the diagram $X_{\mathscr{U}}$. Let X be a topological space and $\mathscr{U} = \{U_a\}_{a\in A}$ an open cover of X. The nerve $N(\mathscr{U})$ has one 0-simplex for each open set in $\mathscr{U}$, whereas $N(X_{\mathscr{U}})$ in addition has 0-simplices for each non-empty intersection $U_{a_1} \cap \ldots \cap U_{a_k}$. It is not hard to convince oneself that all the extra n-simplices added to $N(X_{\mathscr{U}})$ amounts to $N(X_{\mathscr{U}})$ being the *barycentric subdivision* of $N(\mathscr{U})$, as is hinted at in Figure 4.2. Therefore, the simplicial sets $N(\mathscr{U})$ and $N(X_{\mathscr{U}})$ have homotopic geometric realizations,

$$|N(\mathscr{U})| \simeq |N(X_{\mathscr{U}})|.$$

4.4. Mayer-Vietoris blowup of a cover

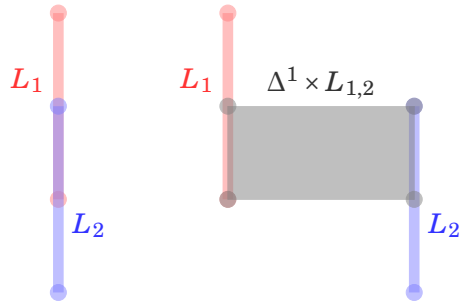
The homotopy colimit of the diagram $X_{\mathscr{U}}$ is called the **Mayer-Vietoris blowup of a cover** $\mathscr{U}$ **of** X. The blowup of a cover will play a central role when studying the Mapper algorithm in Chapter 9, so we present some examples to gain a feel for how the blowup-construction works.

Example 4.4.1. Let $\mathscr{U}$ be the cover $\left\{\left[0,\frac{1}{2}\right],\left[\frac{1}{2},1\right]\right\}$ of the unit interval. The diagram $X_{\mathscr{U}}$ is the "pre-pushout" diagram

$$[0,1/2] \hookleftarrow \{1/2\} \hookrightarrow [1/2,1]$$

and we have already calculated the homotopy colimit of such diagrams in Example 3.4.1. We show the result in Figure 4.3. □

Example 4.4.2. Let X be an interval and $\mathscr{U}$ a cover of X consisting of two subintervals U_1 and U_2 which overlap in more than their two endpoints.



(a) The interval X.

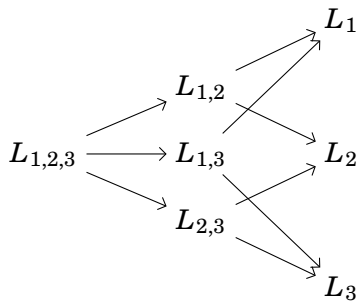
(b) The MV-blowup hocolim$(\mathscr{U})$

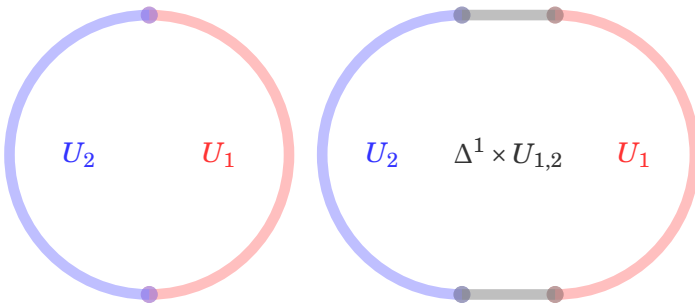
Figure 4.4.: Illustration of Example 4.4.3. The circle X, the cover $\mathscr{U}$ of X consisting of the two half-circles U_1 and U_2 and the Mayer-Vietoris blowup of $\mathscr{U}$, that is, hocolim$(X_{\mathscr{U}})$.

The diagram $X_{\mathscr{U}}$ is once again a "pre-pushout" diagram. We show both the space X and the blowup of $\mathscr{U}$ in Figure 4.4. □

Example 4.4.3. Let X be a circle and $\mathscr{U}$ a cover of X consisting of two half-circles U_1 and U_2 which overlap in their two endpoints. The diagram $X_{\mathscr{U}}$ is also this time a "pre-pushout" diagram. We show both the space X and the blowup of $\mathscr{U}$ in Figure 4.5. □

Example 4.4.4. Let the space X be the three lines L_1, L_2, L_3 shown in Figure 4.6a and let $\mathscr{U}$ be the cover $\{L_1, L_2, L_3\}$ of X. The diagram $X_{\mathscr{U}}$ takes the form





(a) The circle X. (b) The MV-blowup hocolim$(\mathscr{U})$

Figure 4.5.: Illustration of Example 4.4.3. The circle X, the cover $\mathscr{U}$ of X consisting of the two half-circles U_1 and U_2 and the Mayer-Vietoris blowup of $\mathscr{U}$, that is, hocolim$(X_{\mathscr{U}})$.

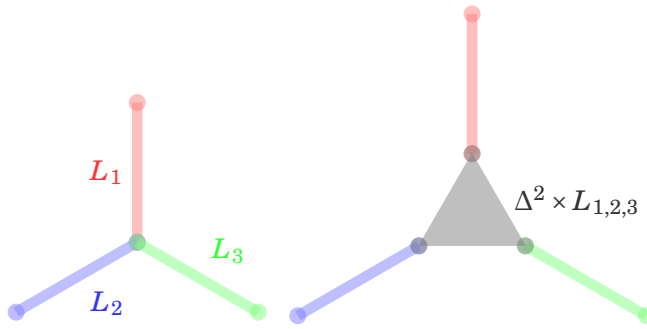
where the spaces $L_{1,2}$, $L_{2,3}$ and $L_{1,2,3}$ all consist of just the middle point of the space X.

We calculate hocolim$(X_{\mathscr{U}})$. After throwing away all degeneracies we are left with the following pieces, which we need to glue together:

$$L_{1,2,3} \times \Delta^2 \coprod L_{1,2} \times \Delta^1 \coprod L_{2,3} \times \Delta^1 \coprod L_{1,3} \times \Delta^1 \coprod L_1 \coprod L_2 \coprod L_3.$$

The edges of L_1, L_2 and L_3 form the boundary of the 1-simplices $L_{i,j} \times \Delta^1$, and the three 1-simplices form the boundary of the 2-simplex. The result of this gluing is shown in Figure 4.6b. □

In all of the above examples the homotopy type of X and hocolim$(X_{\mathscr{U}})$ is the same. This holds in general and will be an important fact in Chapter 9 where we also construct the homotopy equivalence $X \simeq$ hocolim$(X_{\mathscr{U}})$.



(a) The space X. (b) The MV-blowup, $\operatorname{hocolim}(X_{\mathscr{U}})$.

Figure 4.6.: Illustration of Example 4.4.4. The star-shaped space X, the cover $\mathscr{U}$ of X consisting of the three lines L_1 (red), L_2 (blue) and L_3 (green), and the Mayer-Vietoris blowup of $\mathscr{U}$, that is, $\operatorname{hocolim}(X_{\mathscr{U}})$.

5. Clustering

The goal of clustering is to partition data into blocks such that "similar" data is found in the same block.

Example 5.0.5. We present one real-life example of a data set where a cluster analysis seems appropriate. The data set consists of 6 measurements of 200 Swiss bank notes (width of bank note, height on the left side, height on the right side, length of lower margin, upper margin, and inner diagonal), 100 of which are forged and 100 of which are authentic. A successful clustering of these data would cleanly partition the 200 bank notes in two blocks, one block consisting of authentic bills and one block consisting of forged bills. □

It is, at the outset, not clear why a chapter on statistical cluster analysis belongs to a master's thesis on topology. First of all, we need a clustering algorithm as part of the Mapper algorithm of Part II. Secondly, this master's thesis provides two different topological algorithms for data analysis and it is important to determine how these methods compare to other already existing methods, and the only statistical methods which are readily comparable to ours are clustering methods.

We first present clustering methods from the classical statistical point of view before introducing the newer notion of functorial clustering.

5.1. Classic cluster analysis

It is a striking feature of the statistical subfield of cluster analysis that almost no theoretical justification for the different clustering methods exists [12]. This might not have been such a big problem if we had good experimental measures of a clustering algorithm's effectiveness, but no, even measuring how good a clustering is, is a difficult task [12]. A practitioner is therefore urged to "use more than one clustering algorithm" and these

should be “rerun multiple times to find the best possible solution and to test for stability” [6]. That no theoretically justified scheme exists for how one are to go about with this stability testing [2] is peculiar; each practitioner finds his own ad-hoc method for the case at hand. These concerns are reflected in how clustering methods are used; clustering is considered a tool in **exploratory analysis** of the data, that is, the act of getting to know the data on a more intuitive level. Proper hypothesis testing and similar statistical tools might then be used afterwards, on similar but distinct data sets, perhaps on the basis of hypotheses made during the exploratory analysis.

In the following we present some common clustering algorithms to give the reader a feel for the subject and how non-topologists have thought about the clustering problem. There are hundreds of clustering algorithms on the market and since it is impossible to present them all we have chosen three algorithms which are both frequently used and which we feel are representative of a great many other algorithms. The *hierarchical clustering algorithms* represent the most classic algorithms, the *Dbscan algorithm* represents density-based clustering, and *k-means clustering* represents algorithms seeking to find the optimizer of some some real-valued function defined on the set of partitions of the point cloud.

Hierarchical clustering

Hierarchical clustering algorithms construct many clusters which are ordered in a hierarchy with the most granular clustering on the bottom and the clustering consisting of only a single big lump on top. These clustering algorithms come in two types, the **divisive**, which start with the single big lump cluster which is then iteratively divided, and the **agglomerative**, which start with lots of one-point clusters which are then iteratively merged. We will only discuss agglomerative clustering techniques because true divisive algorithms are $\mathcal{O}(2^n)$, which is too slow for practical purposes, whereas some agglomerative algorithms can be made to run in $\mathcal{O}\left(n^2\right)$ time.

The hierarchical clustering methods are built around **linkage functions** which measures the distance between blocks of a partition of a metric space, that is, if (M,d) is a metric space and 2^M is the power set of M, then a linkage function is a function $2^M \times 2^M \to \mathbb{R}_+$.

Such linkage functions give rise to a family of equivalence relations $\sim_{l,\varepsilon}$ on any partition P of M, and we will now construct these $\sim_{l,\varepsilon}$. Let B and B'

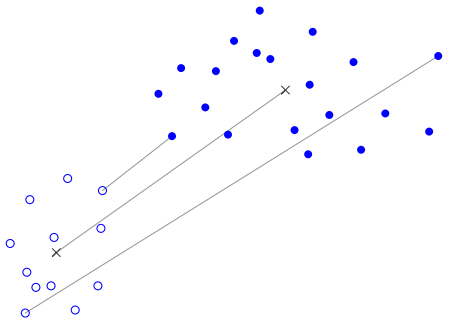


Figure 5.1.: A discrete metric space partitioned in two. The points of one partition are filled and the points of the other partition are not. The lines mark the distance between the partitions as measured by the single-linkage function (shortest), the average-linkage function, and the complete-linkage function (longest).

be two blocks of P, and let ε be a positive real number. The blocks B and B' are equivalent with respect to $\sim_{l,\varepsilon}$ if and only if there in P exists a chain of blocks $B_1,\ldots,B_n$ such that $B_1 = B$, $B_n = B'$ and $l(B_i,B_{i+1}) \leq \varepsilon$ for all the B_i.

We now describe the iterative scheme which defines the agglomerative methods for any finite metric space $M = \{x_0,\ldots,x_n\}$ and any linkage function l. The output of the procedure is a sequence of coarser and coarser partitions $P_0,P_1,\ldots,P_n$ and a corresponding sequence of real numbers $r_0 < r_1 < \ldots < r_n$. Let $P_0 = \{\{x_0\},\ldots,\{x_n\}\}$ be the "discrete" partition of M and let r_0 be zero. Iteratively define r_{i+1} to be $\min\{l(B,B') \mid B,B' \in P_i \text{ and } B \neq B'\}$, and define P_{i+1} to be $P_i/\sim_{l,r_{i+1}}$. That is, P_{i+1} is formed by merging blocks of P_i which lie r_{i+1} away from each other as measured by l.

Common choices for the linkage functions l are

- **single linkage**: $l(B,B') = \min_{x\in B}\min_{y\in B'}\{d(x,y)\}$,
- **complete linkage**: $l(B,B') = \max_{x\in B}\max_{y\in B'}\{d(x,y)\}$,
- **average linkage**: $l(B,B') = \sum_{x\in B}\sum_{y\in B'}\frac{d(x,y)}{\#B\#B'}$, where $\#B$ denotes the number of points in the block B,

which give rise to the algorithms **single linkage clustering**, **complete**

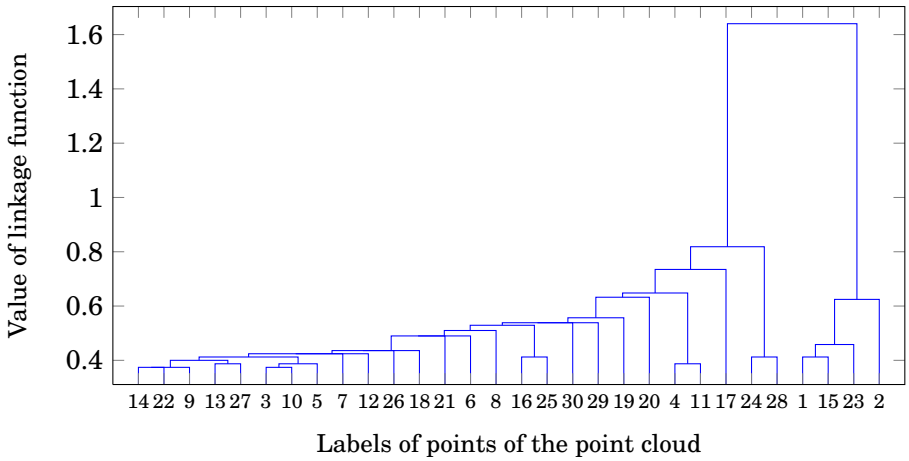


Figure 5.2.: The output of hierarchical clustering algorithms are visualized by dendrograms, as the one shown in this figure. The horizontal line stretching from the mark i to j at height, say r, show that the points x_i and x_j are merged when the linkage function takes the value r. In this particular example it seems clear that the rightmost four points (x_1, x_{15}, x_{23} and x_2) belong to their own cluster and that the rest of the points are more or less clustered together.

linkage clustering and **average linkage clustering** respectively. See Figure 5.1 for a pictorial explanation of the different linkage functions.

Note that we do not get only one clustering of our point cloud M from hierarchical clustering algorithms, we get plenty, namely $P_0,\dots,P_n$. Statisticians have tried to answer the question of which of these clusterings is the best [**?**]. Instead of trying to answer this question, we think of all these $P_0,\dots,P_n$ as informative and "correct", but each at a different scale, the partition P_0 viewing the space at its most granular, P_n viewing the space at its lumpiest, and the rest of the P_i containing distinctions in-between these extremes.

All this information can be visualized by a simple device called a **dendrogram** seen in Figure 5.2. In a dendrogram the vertical axis represent $\mathbb{R}_+$ and on this axis the different values $r_0,\dots,r_n$ have a special meaning since they denote the "time" at which blocks merge. On the horizontal axis

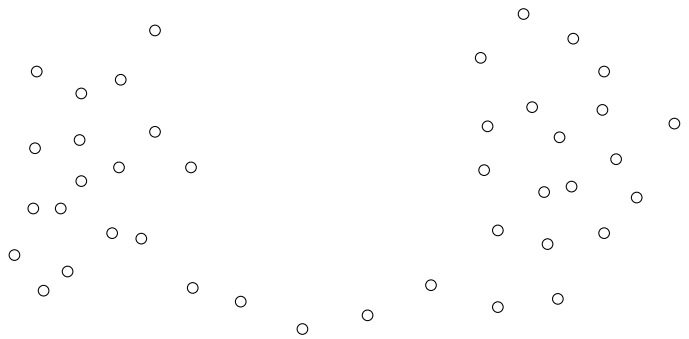


Figure 5.3.: A point cloud which is hard to cluster because of a bridge of connecting points.

the different points $x_1, \ldots, x_n$ are laid out, and if x_i and x_j merge at time r_k, then this is marked by a horizontal line connecting x_i and x_j at height r_k. A picture says more than a thousand words, see Figure 5.2.

DBSCAN

The Dbscan algorithm (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm which uses a measure of density to improve upon linkage clustering. Consider the point cloud in Figure 5.3. It is hard to say what the correct number of clusters in this point cloud is, but it is either one or two. Unfortunately, the only clear answer we will get with linkage clustering is "one", because the two clusters are linked by a thin line of points. The Dbscan algorithm would, at least ideally, give the answer "two" because it does not connect points unless the points are sufficiently "dense".

To define the Dbscan clustering algorithm we need an equivalence relation, which we build by first defining two intermediate relations. Let X be a point cloud, m a natural number and ε a positive real number. We define the function $c_\varepsilon : X \to \mathbb{N}$ to be the function assigning each point $x \in X$ to the number of points of X within a distance ε of x. We say that x is **directly density-reachable (with respect to m and ε)** from y if $d(x,y) \leq \varepsilon$ and $c_\varepsilon(y) \geq m$. This is a reflexive relation on X. We say that x is **density-**

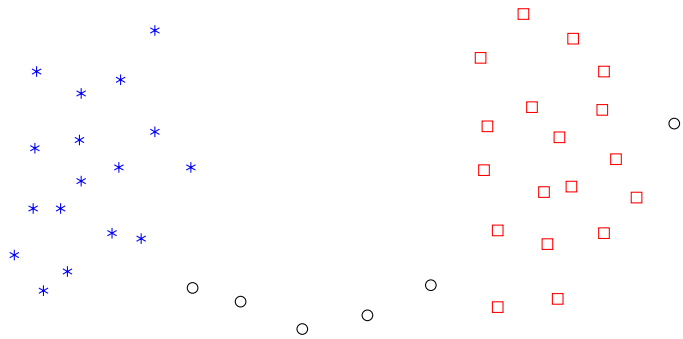


Figure 5.4.: The output of the DBSCAN algorithm used on the point cloud. The asterisks are one cluster and the squares another. The circles are one-point clusters.

reachable from y if there is a sequence of points $p_1, \ldots, p_n$ such that $y = p_1$, $x = p_n$ and p_{i+1} is directly density-reachable from p_i. This is a reflexive and transitive relation on X. Finally, we say that x is **density-connected** to y if there is a point z such that both x and y are density-reachable from z. Density-connectedness is an equivalence relation on X. The Dbscan clustering of X is then defined to be X quotiented out by the density-connectedness equivalence relation.

A characteristic property of the Dbscan clustering is that it returns lots of clusters, most of which contain only a single element. This is so because points which has got no ε-neighbors x with $c_\varepsilon(x) \geq m$ belong to a single-point cluster. This can be considered a way of dealing with noisy data: linkage clustering needs only one point to connect two clusters, whereas Dbscan would need most likely need several.

Note. DBSCAN can easily be extended to give hierarchical output, simply note that for increasing values of ε we get coarser and coarser partitions. This information can then be gathered in a dendrogram, exactly as for linkage clusterings. □

k-means clustering

We now restrict our attention to Euclidean space. Let k be some fixed positive integer, let X a point cloud in $\mathbb{R}^n$ and let $\mathscr{P}_k(X)$ be the family of partitions which partition X into precisely k blocks. The idea behind k-means clustering is to find the partition in $\mathscr{P}_k(X)$ which minimizes the squared distance to block centers, that is, it is the partition $P \in \mathscr{P}_k(X)$ which minimizes the following expression

$$\sum_{B \in P} \sum_{x \in B} |x - \mu_B|^2 \tag{5.1}$$

where μ_B is the centroid of the points in block B.

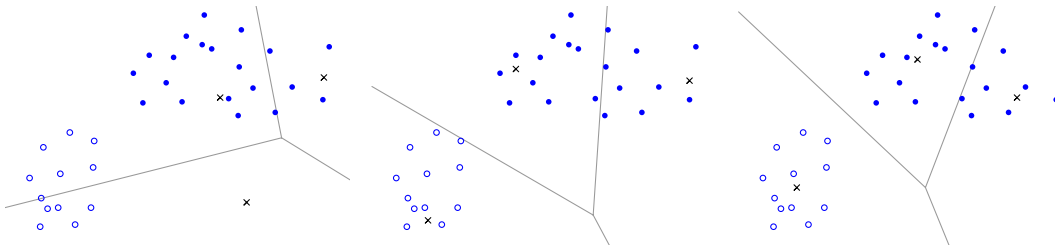
Note. The centroid of a set of points is normally defined only for points in Euclidean space, but we have various ways of simulating centroids also for general metric spaces, and these method extend k-means clustering also to general metric spaces. □

The k-means clustering method has some obvious drawbacks, the biggest of them being that you have to assign the number of clusters k yourself. To decide upon a proper value of k the practitioner is urged to run the algorithm several times for different values of k and either see which answer he likes best or to use one of the many measures for the "goodness" of the clustering. Also, k-means clustering favors clusters which are approximately spherical, since clusters which is very much stretched in some direction give a higher value in expression (5.1).

We need a way to calculate the k-means clustering. Since there are prohibitively many partitions in $\mathscr{P}_k(X)$, we have no way of actually finding the true solution to the k-means clustering, but efficient approximate solvers exist, the following being so ubiquitous that it is simply called *the* k-means clustering algorithm:

The algorithm starts by randomly guessing k cluster centers $c_1^0, \ldots, c_k^0 \in \mathbb{R}^n$ and then iteratively improving these cluster centers until they in some sense correspond to "real" cluster centers. Let $c_1^i, \ldots, c_k^i$ be the cluster centers at the i^{th} timestep and let $V_1^i, \ldots, V_k^i \subseteq \mathbb{R}^n$ be the corresponding Voronoi cells (for their definition, see Section **??**). We then define c_l^{i+1} to be the centroid of the points in $X \cap V_l^i$, that is,

$$c_l^{i+1} = \frac{1}{\#\left(X \cap V_l^i\right)} \sum_{x \in V_l^i} x,$$



(a) Step 1 (b) Step 2 (c) Step 3

Figure 5.5.: The same point cloud as in Figure 5.1, but now used to illustrate the k-means algorithm for $k = 3$. The crosses mark the cluster centroids and the gray lines are the boundaries of the corresponding Voronoi cells. The algorithm converges after only three steps.

where $\#\left(X \cap V_l^i\right)$ denote the cardinality of the set $X \cap V_l^i$. The algorithm converges, and most often does so very quickly, but the worst case running time is exponential.

The algorithm uses random seeds, so running the algorithm several times might be necessary in order to remove random deviations in the solution.

5.2. Some useful categories

An output category for clustering functors

We want a clustering of a point cloud X to contain multi-scale information, so we construct a categorical device to contain such information.

Let X be a finite set and let P and Q be two partitions of X. We say that P **refines** Q if there for each block p of P is a block q of Q such that p is a subset of q. By $\mathscr{P}(X)$ we will mean the category whose objects are all the different partitions of X and where we have an arrow $P \to Q$ when P refines Q. It is easy to check that $\mathscr{P}(X)$ then becomes a well-defined category, the **category of partitions of** X.

We define a **persistent set** to be a pair (X, θ) where X is a finite set and θ is an $\mathbb{R}^+$-persistence object in $\mathscr{P}(X)$ with the additional property that for all s in $\mathbb{R}^+$ there is a real and positive ε such that $\theta(s) = \theta(s')$ for any $s' \in [s, s+\varepsilon]$.

The interpretation is that $\theta(s)$ is a clustering of X at scale s, and that this clustering is stable under slight increases of scale.

Let (X,θ) and (Y,η) be persistent sets and f a map $X \to Y$. Note that a partition $P = \{P_i\}_{i\in A}$ of the finite set Y induces a partition $\left\{f^{-1}(P_i)\right\}_{i\in A}$ of X, and we denote this partition by $f^{-1}(P)$. We say that a function f is **persistence preserving** if $\theta(s)$ refines $f^{-1}\left(\eta(s)\right)$ for any scale s.

We can now readily check that we have a **category of persistent sets and persistence preserving maps**, which we will denote by $\underline{\mathscr{P}}$; indeed, the identity morphisms are persistence preserving and the composition of two persistence preserving maps is persistence preserving.

Metric space categories

Our input category for clustering functors should be a category of (finite) metric spaces. In any category of metric spaces it is obvious what the objects should be (they should be metric spaces), but we have many different notions of morphisms to choose between.

Let (M,d_M) and (N,d_N) be metric spaces. **Isometries** are functions $f : (M,d_M) \to (N,d_N)$ such that $d_M(x,y) = d_N(f(x),f(y))$ for all x and y in M. We denote the category of metric spaces with isometries as morphisms by $\mathscr{M}^{\text{iso}}$.

Weak contractions (or **distance non-increasing maps**) are functions $f : (M,d_M) \to (N,d_N)$ such that $d_M(x,y) \leq d_N(f(x),f(y))$ for all x and y in M. Note that isomorphisms in this category are bound to be isometries. We denote the category of metric spaces with weak contractions as morphisms by $\mathscr{M}^{\text{gen}}$. (We choose the notation $\mathscr{M}^{\text{gen}}$ only to comply with the notation in [5]; we do not know what "gen" stands for or why this name is chosen.)

Monic maps are weak contractions which are inclusions on the underlying sets, and we denote the category of metric spaces and monic maps with $\mathscr{M}^{\text{mon}}$.

These categories are related by the following chain of faithful (but not full) inclusions of categories

$$\mathscr{M}^{\text{iso}} \subseteq \mathscr{M}^{\text{mon}} \subseteq \mathscr{M}^{\text{gen}}.$$

Note. One obvious choice of morphisms for a category $\mathscr{M}$ of metric spaces is continuous functions, but since we are dealing with discrete metric spaces all functions are continuous and it is too strict to be a functor from such a

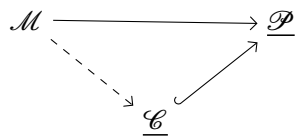
category. For example, we should not expect to find a clustering functor $\mathfrak{C}$ inducing a function $\mathfrak{C}(f)$ between clusters for a function f which redistributes the points of our point cloud wildly. We therefore need restrictions on our choice of morphisms and this disqualifies continuous functions. □

5.3. Functorial clustering

We are now ready to define what we mean by functorial clustering. Let $\mathscr{M}$ be either $\mathscr{M}^{\text{iso}}$, $\mathscr{M}^{\text{gen}}$ or $\mathscr{M}^{\text{mon}}$. A **functorial hierarchical clustering algorithm** is a covariant functor

$$\mathscr{M} \to \underline{\mathscr{P}}.$$

This definition might seem to restrictive because there are so many clustering algorithms which do not output the kind of hierarchical structure present in persistent sets, but these correspond to the persistent sets with *trivial* hierarchical structure. More formally, we say that the persistent set (X,θ) is **constant** if $\theta(s) = \theta(s')$ for all scales s and s'. The constant persistent sets form their own category $\underline{\mathscr{C}}$ and we say that a functorial clustering algorithm $\mathscr{M} \to \underline{\mathscr{P}}$ is a **standard functorial clustering algorithm** if it factors through the inclusion $\underline{\mathscr{C}} \hookrightarrow \underline{\mathscr{P}}$:



Equivalently, we can define a standard functorial clustering algorithm to be a covariant functor $\mathscr{M} \to \underline{\mathscr{C}}$. This being said, in this thesis we will mostly be discussing hierarchical clustering algorithms.

Note. A functorial clustering algorithm on $\mathscr{M}^{\text{gen}}$ is also a functorial clustering algorithm on $\mathscr{M}^{\text{mon}}$, and a functorial clustering algorithm on $\mathscr{M}^{\text{mon}}$ is also a functorial clustering algorithm on $\mathscr{M}^{\text{iso}}$, that is, as we allow more morphisms into the category we disallow some clustering algorithms on the category. To see this, note that there are many more morphisms in $\mathscr{M}^{\text{gen}}$ than in $\mathscr{M}^{\text{mon}}$ and $\mathscr{M}^{\text{iso}}$,

$$\hom_{\mathscr{M}^{\text{iso}}}(X,Y) \subseteq \hom_{\mathscr{M}^{\text{mon}}}(X,Y) \subseteq \hom_{\mathscr{M}^{\text{gen}}}(X,Y),$$

so if a clustering algorithm $\mathfrak{C}$ is functorial on all the morphisms in $\mathscr{M}^{\text{gen}}$, then $\mathfrak{C}$ is automatically functorial on the morphisms on $\mathscr{M}^{\text{mon}}$ and $\mathscr{M}^{\text{iso}}$. □

We now redefine the clustering algorithms of Section 5.1 in terms of the categories of Section 5.2. That is, we will present how a hierarchical clustering algorithm in a natural way gives us an assignment of objects and morphisms $\mathscr{M} \to \underline{\mathscr{P}}$.

Let X be a point cloud. Recall that a hierarchical clustering of X is a sequence of partitions $P_0, \ldots, P_n$ of X and a corresponding sequence of positive real numbers $r_0, \ldots, r_n$. Define $\theta_X : \mathbb{R}_+ \to \mathscr{P}(X)$ to be the $\mathbb{R}_+$-persistence object which assigns s to the partition P_i if $r_i \leq s < r_{i+1}$. One way to think of $\theta_X(s)$ is as the single linkage partition of X at scale s. We can now define a function from $\text{Ob}(\mathscr{M})$ to $\text{Ob}\left(\underline{\mathscr{P}}\right)$ by $X \mapsto (X, \theta_X)$ and a function from $\hom_{\mathscr{M}}(X,Y)$ to $\hom_{\underline{\mathscr{P}}(X)}((X,\theta_X),(Y,\theta_Y))$ by sending a morphism $f : X \to Y$ to the morphism $\tilde{f} : (X,\theta_X) \to (Y,\theta_Y)$ whose action is to apply f on the underlying set X of (X,θ_X).

This procedure produces an assignment of objects and morphisms for both the linkage algorithms and Dbscan, but we have not yet made the claim that this assignment is a functor. It turns out that some clustering algorithms are not functorial at all, that other clustering algorithms are functorial only on some categories of metric spaces, and that some clustering algorithms are functorial on all our categories of metric spaces. We will now describe whether or not the linkage algorithms and Dbscan are functorial on $\mathscr{M}^{\text{iso}}$, $\mathscr{M}^{\text{mon}}$ and $\mathscr{M}^{\text{gen}}$, but first, we look at k-means clustering.

Example 5.3.1. The k-means clustering algorithm is not functorial in any of the metric spaces categories since it is dependent on information not contained in the metric. For instance, k-means clustering uses randomized seeds and might converge only to a local optimum. □

The rest of our algorithms are functorial with respect to $\mathscr{M}^{\text{iso}}$ by the following argument: since isometries preserve the metric space structure, being functorial with respect to these maps only means that the clustering algorithm does not use any extraneous information when building the clusters. Therefore, except for k-means clustering, all our algorithms are $\mathscr{M}^{\text{iso}}$-functorial.

Proposition 5.3.1. *Complete linkage and average linkage clustering are not functorial on $\mathscr{M}^{mon}$ (and therefore also not on $\mathscr{M}^{gen}$).*

Proof. We construct a counter-example showing that complete linkage clustering and average linkage clustering are not functorial on $\mathscr{M}^{\text{mon}}$.

Let us call the persistent set built from complete linkage clustering ψ_X and the resulting clustering algorithm $\mathfrak{C}$. Now consider the two graphs $X = \{A,B,C\}$ and $Y = \{A',B',C'\}$ shown in Figure 5.6 and the graph-morphism $f : X \to Y$ defined by $f(A) = A'$, $f(B) = B'$ and $f(C) = C'$. It is clear how X and Y can be regarded as metric spaces and f as a metric space morphism. Directly below the graphs X and Y are the dendrograms given by the complete linkage clustering ψ_X and ψ_Y. For $\mathfrak{C}$ to be functorial $\psi_X(s)$ must refine $f^{-1}\big(\psi_Y(s)\big)$ for all $s \in \mathbb{R}_+$, but this is not the case because $\psi_X(2) = \{\{A\},\{B,C\}\}$ and $f^{-1}\big(\psi_Y(2)\big) = \{\{A,C\},\{B\}\}$.

The function f also shows that average-linkage fails to be functorial, but we do not provide the details. □

From the counter-example in the proof we extract the following morale: Let $\mathfrak{C}$ be any hierarchical clustering algorithm on $\mathscr{M}^{\text{iso}}$. If there exists a space X with points $x, y \in X$ such that for a scale s the algorithm $\mathfrak{C}$ clusters x and y together, but which for a contraction f does *not* cluster $f(x)$ and $f(y)$ together, then we can construct a counter-example as in the preceding proof.

Proposition 5.3.2. *DBSCAN clustering is functorial on $\mathscr{M}^{mon}$.*

Proof. It is quite easy to check that DBSCAN is functorial on $\mathscr{M}^{\text{mon}}$ and the argument is similar to the one found in the proof of Proposition 5.3.4. □

Proposition 5.3.3. *DBSCAN clustering is not functorial on $\mathscr{M}^{gen}$.*

Proof. We give a counter-example showing that DBSCAN is not functorial on $\mathscr{M}^{\text{gen}}$.

Consider the metric spaces $X = \{A,B,C\}$ and $Y = \{A',B'\}$ and the metric space morphism f given by $f(A) = A'$, $f(B) = B'$ and $f(C) = B'$. We do DBSCAN clustering based on the values $m = 3$ and $\varepsilon = 1$, that is, three points are in the same cluster if they are within a distance 1 of each other. Hence, $\text{Dbscan}(X,m,\varepsilon) = \{\{A,B,C\}\}$, but $\text{Dbscan}(Y,m,\varepsilon) = \big\{\{A'\},\{B'\}\big\}$ so we see that $\text{Dbscan}(X,m,\varepsilon)$ does not refine $f^{-1}(\text{Dbscan}(Y,m,\varepsilon)) = \{\{A\},\{B,C\}\}$. □

We can extract the following morale from the counter-example in the proof of Proposition 5.3.2: Density based clustering algorithms $\mathfrak{C}$ can never be functorial on $\mathscr{M}^{\text{gen}}$ since the collapsing of points leads to a less dense metric

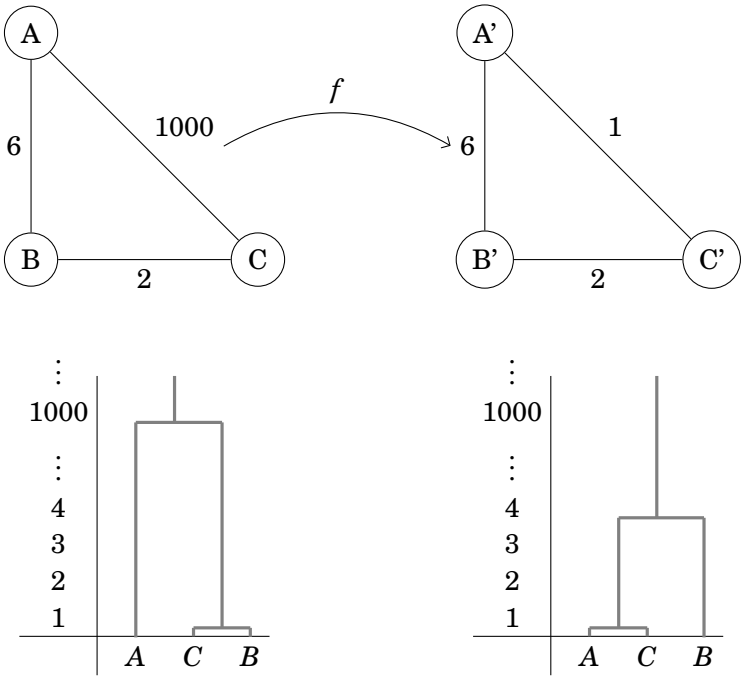


Figure 5.6.: Counter-example showing that complete and average linkage clustering are not functorial on $\mathscr{M}^{\text{mon}}$. The graphs are metric spaces with distances given by the edge weights and f is a morphism in $\mathscr{M}^{\text{mon}}$. Directly below are the dendrograms given by complete linkage clustering. At height 2 the right dendrogram does not refine the left one and this provides the counterexample.

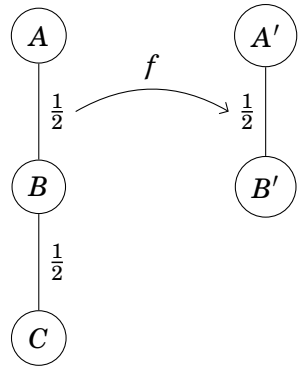


Figure 5.7.: Counter-example showing that Dbscan is not functorial on $\mathscr{M}^{\text{gen}}$. The Dbscan clustering with parameters $m = 3$ and $\varepsilon = 1$ of the right metric space is $\{\{A'\},\{B'\}\}$ and of the left metric space is $\{\{A\},\{B\},\{C\}\}$.

space, and less dense metric spaces lead to a more granular clustering under $\mathfrak{C}$.

We denote the assignment obtained from single linkage clustering by $\mathfrak{R}$.

Proposition 5.3.4. *The **single linkage clustering** $\mathfrak{R} : \mathscr{M}^{gen} \to \underline{\mathscr{P}}$ is a functor on $\mathscr{M}^{gen}$ (and hence also on $\mathscr{M}^{mon}$ and $\mathscr{M}^{iso}$).*

Proof. We need to prove that $\mathfrak{R}$ produces well-defined objects and well-defined morphisms, and that the morphisms adhere to the composition rules and the identity.

The object $\mathfrak{R}(X) = \theta_X$ is clearly a well-defined object in $\underline{\mathscr{P}}$ since $\theta_X(s)$ refines $\theta_X(s')$ when $s \leq s'$.

We will need the following observation: $\theta_X(s)$ is nothing more than $X/{\sim_s}$ where the equivalence relation $\sim_s$ is such that two points x and x' are equal if and only if there is a chain of points $x_0,\dots,x_k \in X$ such that $x_0 = x$, $x_k = x'$ and $d_X(x_i, x_{i+1}) \leq s$.

To show that $\mathfrak{R}(f) : \mathfrak{R}(X) \to \mathfrak{R}(Y)$ is a morphism in $\underline{\mathscr{P}}$ we need to show that $\theta_X(s)$ refines $f^{-1}(\theta_Y(s))$ and for this to be true it suffices to show that for a block $\{x_1,\dots,x_k\}$ of $\theta_X(s)$ the points $f(x_1),\dots,f(x_k)$ are contained in the same block of $\theta_Y(s)$. But since the points $\{x_1,\dots,x_k\}$ are in the same block they are equivalent under $\sim_s$, and since f is a weak contraction, the points $f(x_1),\dots,f(x_k)$ are also equivalent under $\sim_s$ and hence lie in the same block.

That this assignment of morphisms is functorial follows immediately. □

Part II.

Method

6. Mapper

The Mapper algorithm is a tool for analyzing high-dimensional data sets or data sets in arbitrary metric spaces.

The algorithm has recently proved useful in applications; a new genetically distinct subspecies of breast cancer was identified using Mapper [14], and folding pathways of proteins have been studied [18].

The inventors of the algorithm have started a firm, Ayasdi, and the interested reader is urged to visit `www.ayasdi.com` to see how a group of researchers at Stanford intend to sell topology on the private market.

We will first motivate and present the main ideas behind the algorithm before giving details. Afterwards we connect these ideas with standard topological machinery, functoriality and the homotopy colimit. The reader only need to read through Chapter 7 before being able to understand some of the content of the experiments in Part III, and it might then be beneficial to read ahead some to get a feel for how Mapper is used before delving into the theoretical considerations of Chapter 8.

There is really only one published article devoted to Mapper, "Topological Methods for the Analysis of High Dimensional Data Sets and 3D Object Recognition" by Gurjeet Singh, Facundo Mémoli and Gunnar Carlsson [17], but Mapper is also briefly described in Carlsson's survey article "Topology and data" [3]. My addition to this literature is to write down an account of how Mapper is a functor and of how Mapper relates to homotopy colimits.

6.1. Motivating example

A common way of studying a topological space X is to examine a continuous mapping $f : X \to Z$ from X to some well-understood topological space Z. Through characteristics of the space Z and the map f we hope to recover information about X. This is indeed the strategy we are going to use now; we want to obtain open covers of X by pulling back open covers of Z with f

and see what information about X we recover from the nerve of these covers. We will call the function f a **filter function** and the space Z together with an open cover a **parameter space**.

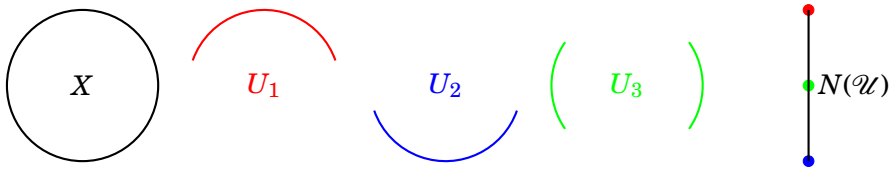
We illustrate the process with an example. Let $\mathbb{S}^1$ be the 1-sphere realized as the circle of radius 1 centered at the origin of $\mathbb{R}^2$, and let the parameter space be the interval $[-1,1]$. We define the filter function $f : \mathbb{S}^1 \to [-1,1]$ to be the function projecting $\mathbb{S}^1$ onto its y-coordinate. We choose an open cover $\mathscr{I}$ of the parameter space consisting of the intervals

$$I_1 = (0.33, 1],\ I_2 = [-1, -0.33),\ I_3 = (-0.5, 0.5).$$

This cover induces a cover $\mathscr{U} = f^{-1}(\mathscr{I})$ of $\mathbb{S}^1$, consisting of the sets

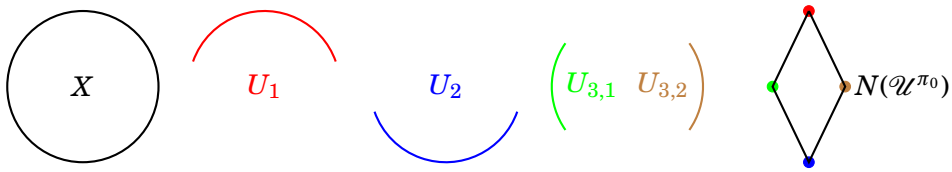
$$U_1 = f^{-1}(I_1),\ U_2 = f^{-1}(I_2),\ U_3 = f^{-1}(I_3).$$

The space $\mathbb{S}^1$, the cover $\mathscr{U}$ and the nerve $N(\mathscr{U})$ is depicted below.



Note that the nerve $N(\mathscr{U})$ does not have the same homotopy type as $\mathbb{S}^1$, so, in our attempt at approximating the space $\mathbb{S}^1$ with the nerve of an open cover, we failed.

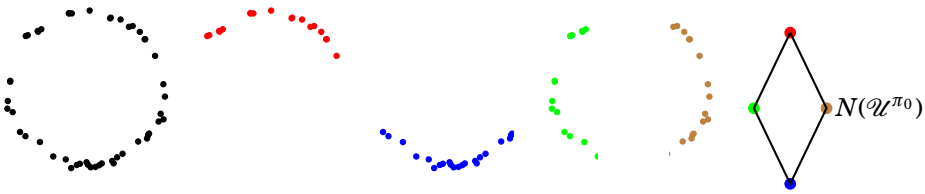
We can fix this discrepancy in homotopy type by defining a new cover $\mathscr{U}^{\pi_0} = \{U_1, U_2, U_{3,1}, U_{3,2}\}$ where $U_{3,1}$ and $U_{3,2}$ denote the two connected components of U_3. (We choose the superscript π_0 to remind us of connected components.) The new cover is shown in the following picture.



Note that, this time, the homotopy type of $N(\mathscr{U}^{\pi_0})$ and $\mathbb{S}^1$ is the same.

We have just gone through all the steps that constitute the Mapper algorithm. In short, Mapper is nothing more than the observation that a filter $f : X \to Z$ induces covers $\mathscr{U}$ whose nerve can be used to study X, but with the important refinement that one first splits the open sets of $\mathscr{U}$ into its connected components, thereby creating $\mathscr{U}^{\pi_0}$.

We will mostly be concerned with the analysis of point clouds, and for point clouds the notion of a "connected component" should be replaced with "cluster of points. That is, in order to apply Mapper to point cloud data you must first obtain a cover $\mathscr{U}$ induced by some filter function. You then create $\mathscr{U}^{\pi_0}$ by *clustering* each set of the cover and then construct $N(\mathscr{U}^{\pi_0})$ from these clusters. The picture is completely analogous.



We now turn to the formalization of this process.

7. Explicit description of Mapper

The Mapper algorithm comes in both a topological version and in a statistical version. We saw both flavors in the last section. In the next two sections we describe these two versions of Mapper and give examples of how they work.

7.1. The topological Mapper

Let X be a topological space, let Z with its open cover $\mathscr{U} = \{U_\alpha\}_{\alpha \in A}$ be a parameter space, and let $f : X \to Z$ be a filter function.

The filter f induces an open cover of X from $\mathscr{U}$, as explained in Chapter 4. We denote this induced cover on X by $f^{-1}(\mathscr{U})$ and its open sets by U'_α. Each of these U'_α might consist of several connected components, even when all the U_α consist of only one connected component. Let $n_\alpha \in \mathbb{N}$ be the number of connected components of U'_α and let $U'_{\alpha,1}, \ldots, U'_{\alpha,n_\alpha}$ denote the connected components of U'_α. We then define the open cover $f^{-1}(\mathscr{U})^{\pi_0}$ to be

$$\left\{ U'_{\alpha,i} \;\middle|\; \text{for all } \alpha \in A \text{ and } i = 1, \ldots, n_\alpha \right\}.$$

The nerve $N\left(f^{-1}(\mathscr{U})^{\pi_0}\right)$ is the output of the Mapper algorithm and we will say that $N\left(f^{-1}(\mathscr{U})^{\pi_0}\right)$ is **Mapper of X with filter f and cover $\mathscr{U}$**. We write this as $\mathit{Mapper}\left(f^{-1}(\mathscr{U})\right)$.

Example 7.1.1. In the introduction we saw Mapper used on the circle $\mathbb{S}^1$. Now, in the same vein, we study $\mathbb{S}^2$ realized as the 2-sphere of radius 1 in $\mathbb{R}^3$ centered in the origin. Let $f : \mathbb{S}^2 \to [-1,1]$ be the projection onto one of the axes and let $\mathscr{U}$ be the cover $\{[-1,-0.33),(-0.5,0.5),(0.33,1]\}$ of $[-1,1]$. We see that the cover $f^{-1}(\mathscr{U})$ of $\mathbb{S}^2$ consists of the upper cap of $\mathbb{S}^2$, the lower cap of $\mathbb{S}^2$ and a belt in the middle; and that $\mathit{Mapper}\left(f^{-1}(\mathscr{U})\right)$ is the simplicial complex •——•——•. So in this case, Mapper fails to preserve the homotopy type of the space. □

Example 7.1.2. Recall the example given in Section 6.1 with $X = \mathbb{S}^1$, $Z = [-1,1]$, $\mathscr{U} = [-1,-0.33],[-0.5,0,5],[0.33,1]$ and f the projection onto the y-axis. Mapper then produced a graph Γ which was homotopy equivalent to $\mathbb{S}^1$. We now take X to be the torus $\mathbb{T}$ realized around the origin in $\mathbb{R}^3$, let f still be the projection onto the y-axis, and take the same cover $\mathscr{U}$ of $[-1,1]$. A little thought shows that Mapper reproduces the same graph Γ. In fact, the projection down to any axis with similar covers will reproduce the graph Γ or a graph homotopy equivalent to Γ.

Thus Mapper once again fails to recover the homotopy type of the space. □

Example 7.1.3. Let X be any finite metric space and let n be the number of points of X. No matter what filter f or parameter space $(X,\mathscr{U})$ we choose, the output of the *topological* Mapper will have n connected components and be totally disconnected. Mapper produces this output because the connected components of any set $U \subseteq X$ are the points of U itself, therefore $\mathscr{U}^{\pi_0}$ will be a trivial cover for all covers $\mathscr{U}$ of X. □

None of these examples are especially interesting, after all, we have much better methods for studying general topological spaces, but the examples serve to illustrate how the method works for the case we do not understand so well, the point cloud case.

7.2. The statistical Mapper

The statistical Mapper is very much analogous to the topological Mapper, the only difference being that, in the statistical Mapper, the notion of a connected component is replaced by the notion of a cluster. These notions are related in the following way; the connected components of a topological space X is a partitioning of X, and a clustering of a point cloud Y is a partitioning of Y, such that, had the point cloud Y been sampled from the topological space X, the clustering of Y would have corresponded to the different connected components of X. Even though the statistical and topological Mapper in this way are very similar, we choose to spell out the statistical Mapper too. At this stage it does not matter exactly which clustering algorithm we choose, so simply assume that we have some standard clustering algorithm, and call it $\mathfrak{C}$.

Let X be a finite metric space, let Z be a topological space and let $f : X \to Z$ be a function. A cover $\mathscr{U} = \{U_\alpha\}_{\alpha \in A}$ of Z induces a cover $f^{-1}(\mathscr{U})$ of X and we denote the sets of $f^{-1}(\mathscr{U})$ by U'_α. Each U'_α consists of many connected components, in fact, each point of U'_α is itself a connected component. Thus, partitioning U'_α into its connected components, as we would have done with the topological Mapper, will not help us any further, so at this point we turn to partitioning by clustering. The clustering algorithm $\mathfrak{C}$ partitions U'_α in, say, n_α sets, which we denote by $U'_{\alpha,1}, \ldots, U'_{\alpha,n_\alpha}$. Then we define the cover $f^{-1}(\mathscr{U})^{\mathfrak{C}}$ of X to be

$$\left\{ U'_{\alpha,i} \;\middle|\; \text{for all } \alpha \in A \text{ and } i = 1, \ldots, n_\alpha \right\}.$$

The nerve $N\left(f^{-1}(\mathscr{U})^{\mathfrak{C}}\right)$ is the output of the statistical Mapper algorithm and we will say that $N\left(f^{-1}(\mathscr{U})^{\mathfrak{C}}\right)$ is **Mapper of X with filter f, cover $\mathscr{U}$ and clustering $\mathfrak{C}$**. We write this as $Mapper\left(f^{-1}(\mathscr{U}), \mathfrak{C}\right)$.

7.3. Filters and parameter spaces

We describe some filter-functions and the properties we expect them to have in applications.

Eccentricity

The *eccentricity* of a point x in a point cloud X is a measure of how close the point X lies to the "center" of the point cloud. Points lying far away from the center get high values of eccentricity and vice versa. Point clouds do not come with a predefined center, so we have to do with an approximate measure. We define the n^{th} **eccentricity** of the point x in X to be

$$ecc_n(x) = \left(\sum_{y \in X} d(x,y)^n \right)^{1/n},$$

where $d : X \times X \to \mathbb{R}$ is the metric of the point cloud X.

Density

The *density* of a point x of a point cloud X is a measure of how close x are to surrounding points. Points which lie far away from their closest neighbors

get low density values and vice versa. There are many ways of measuring density, and the interested reader will find plenty in an appropriate book or survey article, but we shall describe only one, since density estimation plays only a very minor role in this document.

The k^{th} **nearest neighbor density** of a point x of X is

$$density_k(x) = \frac{k}{|X| d_k(x)}$$

where $|X|$ is the number of points in X and $d_k : X \to \mathbb{R}$ is the distance to the k^{th} nearest neighbor of x.

Mixed filters

Although we have presented only two filter functions (and the original article [17] there was just one more, the *graph Laplacian*) we can create some more through combinations of these.

One way of combining filters is with the tuple $(ecc, density) : X \to \mathbb{R}^2$ defined as $(ecc, density)(x) = (ecc(x), density(x))$. This is a more sensitive filter than either *ecc* or *density* alone, since points which have the same eccentricity can have different density and conversely.

Another way is with the ordinary arithmetic operations, one can for example get the filter $f_+ : X \to \mathbb{R}$ defined as $f_+(x) = ecc(x) + density(x)$, and similarly for $-$, $\cdot$ and $\div$. With these one can filter on the different combinations of high/low density and high/low eccentricity.

We now turn to some parameter spaces. We restrict ourselves to $\mathbb{R}^n$ and compact subsets $\mathbb{R}^n$, although any topological space X for which we have some cover $\mathscr{U}$ can be used as a parameter space.

A family of covers of an interval $[a, b]$

We now describe a very useful class of covers of the closed interval $[a, b] \subseteq \mathbb{R}$. These covers are made from n intervals of equal length $l = \frac{b-a}{n}$, where p percent of an interval overlaps with the subsequent interval. We call this cover $\mathscr{U}(n, p)$.

Example 7.3.1. For the choice of parameters $[a, b] = [0, 1]$, $n = 2$ and $p = \frac{1}{2}$ we get the cover

$$\left\{ [0, \frac{2}{3}), (\frac{1}{3}, 1] \right\}.$$

The length l of each interval is $\frac{2}{3}$ and the length of the overlap is $\frac{1}{3}$, such that $p = \frac{1}{2}$ is the overlap percent, exactly as wanted. □

We explain how this cover is used in applications: Choose a filter function $f : X \to \mathbb{R}$ on your point cloud X and calculate the filter value of all points $x \in X$. Set a to be the minimum filter value and set b to be the maximum filter value. Choose different values for n and p, create the associated cover $\mathscr{U}(n,p)$ of $[a,b]$ and examine the Mapper output,

$$Mapper\left(f^{-1}(\mathscr{U}(n,p))\right).$$

If the Mapper output is too granular or too lumpy, repeat the process for other choices of p and n. Lower values of p make for less overlapping covering sets and hence a less connected Mapper output, while lower values of n make for fewer and bigger covering sets and hence fewer nodes in the Mapper output.

We can describe a better way of choosing parameters p and n after having proved that Mapper is a functor, as functoriality opens up for the use of the ideas of *persistence*.

A family of covers of n-orthotopes

We have a similar class of covers of $\prod_{i=1}^{N}[a_i,b_i] \subseteq \mathbb{R}^N$ as we had for $[a,b] \subset \mathbb{R}$. For $i = 1,\ldots,N$ let $\mathscr{U}^i = \{U_1^i,\ldots,U_{n_i}^i\}$ be a cover of $[a_i,b_i]$ consisting of n_i intervals of the kind described in the previous subsection. We define a cover of $\prod_{i=1}^{N}[a_i,b_i]$ to be

$$\left\{\prod U_\alpha^i \times U_\beta^j \;\middle|\; \text{for all } i,j = 1,\ldots,N, \text{ with } i \neq j \text{ and } \alpha = 1,\ldots,n_i \text{ and } \beta = 1,\ldots,n_j\right\}$$

8. Categorical description of Mapper

We now want to describe Mapper as an algorithm on diagrams $X_{\mathcal{U}}$. This description of Mapper makes it easier to formulate and to prove various properties of the algorithm.

8.1. The topological Mapper

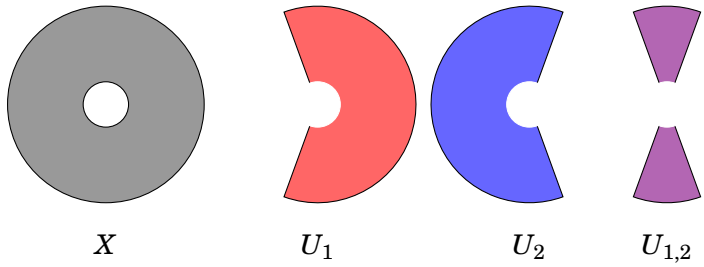
Mapper consists of two "parts"; splitting a cover into connected components and taking the nerve. We know how to take the nerve of a diagram, so the only part we need to describe is how to mirror on diagrams $X_{\mathcal{U}}$ the process of going from $\mathcal{U}$ to $\mathcal{U}^{\pi_0}$. That is, we now describe a new diagram $\pi_0(X_{\mathcal{U}})$ which corresponds to $\mathcal{U}^{\pi_0}$. This assignment $\pi_0(-)$ works on any object of the functor category Top^{Cat}, so we choose to formulate π_0 on general diagrams and not just on diagrams of the form $X_{\mathcal{U}}$ for a space X and cover $\mathcal{U}$.

Let $D\colon A \to Top$ be an object of Top^{Cat}. We now describe the diagram $\pi_0(D)$. The process is illustrated in Figure 8.1. Let D_α be a space of the diagram D and denote the connected components of D_α by $D_{\alpha,i}$ where i is an element of the index set I_α. We define a new diagram $\pi_0(D)\colon A' \to Top$, where A' is a partially ordered set with elements (α, i) for $\alpha \in A$ and $i \in I_\alpha$ and where we have an A'-morphism $(\alpha, i) \leq (\beta, j)$ if we have a map $f_\alpha\colon D_\alpha \to D_\beta$ in D satisfying $f_\alpha(D_{\alpha,i}) \subseteq D_{\beta,j}$. We then define the component of the natural transformation f at (α, i), the map $f_{\alpha,i}\colon D_{\alpha,i} \to D_{\beta,j}$, to be the restriction of f_α to $D_{\alpha,i}$.

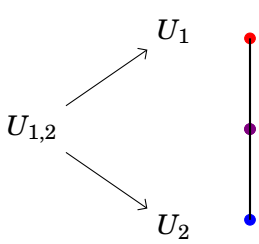
Let $\mathcal{U}$ be a cover of the topological space X. We now define $Mapper(\mathcal{U})$ to be the composition

$$|-| \circ N \circ \pi_0 \circ \mathfrak{D}(\mathcal{U}),$$

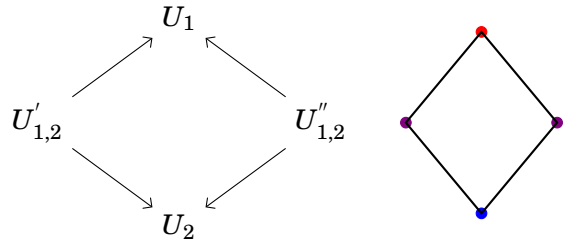
where we remember from Section 4.2 that $\mathfrak{D}$ was the functor assigning $\mathcal{U}$ to $X_{\mathcal{U}}$.



(a) Space X and cover $\mathcal{U} = \{U_1, U_2\}$ and intersection $U_{1,2}$



(b) The diagram $X_{\mathcal{U}}$ and its nerve



(c) The diagram $\pi_0(X_{\mathcal{U}})$ and its nerve

Figure 8.1.: A space X, a cover $\mathcal{U} = \{U_1, U_2\}$ and the diagrams $X_{\mathcal{U}}$ and $\pi_0(X_{\mathcal{U}})$. In the diagram $\pi_0(X_{\mathcal{U}})$ the intersection $U_{1,2}$ has been split into its connected components $U'_{1,2}$ and $U''_{1,2}$ and the appropriate inclusion morphisms have been added.

8.2. The statistical Mapper

Let $\mathcal{M}$ be any of the metric space categories $\mathcal{M}^{\text{iso}}$, $\mathcal{M}^{\text{mon}}$ or $\mathcal{M}^{\text{gen}}$ and let $\mathfrak{C}$ be any standard (not necessarily functorial) clustering algorithm on $\mathcal{M}$.

In the last subsection we built a diagram $\pi_0(D)$ from some diagram D of topological spaces. In a completely analogous fashion, we now construct the diagram $\mathfrak{C}(D)$ from some diagram D of finite spaces of $\mathcal{M}$.

We define the action of $\mathfrak{C}$ on an object $D : A \to Top$ of $\mathcal{M}^{Cat}$. Denote by $D_{\alpha,i}$ the different clusters of $\mathfrak{C}(D_\alpha)$ where i is an element of an index set I_α. We define the new diagram $\mathfrak{C}(D) : A' \to \mathcal{M}$ where A' is a partially ordered set with with elements (α, i) for $\alpha \in A$ and $i \in I_\alpha$ and where we have an A'-morphism $(\alpha, i) \leq (\beta, j)$ if we have a map $f : D_\alpha \to D_\beta$ in D satisfying

$f\left(D_{\alpha,i}\right) \subseteq D_{\beta,j}$. We then define the component of f at (α, i), $f_{\alpha,i} : D_{\alpha,i} \to D_{\beta,j}$, to be the restriction of f_α to $D_{\alpha,i}$.

Let $\mathscr{U}$ be a cover of a finite metric space M. We define $Mapper(\mathscr{U}, \mathfrak{C})$ to be the composition

$$|-| \circ N \circ \mathfrak{C} \circ \mathfrak{D}(M, \mathscr{U}).$$

8.3. Functoriality of Mapper

As we will show in this section, both the statistical and topological Mapper algorithms are functors. This functoriality is now quite immediate and might seem obvious, but we believe that this apparent simplicity is due to our choice of background material and the way we have defined Mapper in Chapter 8, which is different from the standard way presented in [17] and [4] and Chapter 7. At least, to the best of the author's knowledge, there is no mention in the literature of Mapper being a functor and none of the immediate ideas following from functoriality has so far been exploited.

Functoriality of the topological Mapper

To prove that the topological Mapper is a functor we extend the assignment π_0, which so far is an assignment on only the objects of Top^{Cat}, to also be an assignment on morphisms of Top^{Cat}. When we have proved that π_0 is a functor we see that Mapper is a functor because Mapper is, by the definition given in Chapter 8, defined as the composition

$$TopCov \xrightarrow{\mathfrak{D}} Top^{Cat} \xrightarrow{\pi_0} Top^{Cat} \xrightarrow{N} sSet \xrightarrow{|-|} Top,$$

and all arrows in this chain are functors.

We now define π_0 on Top^{Cat}-morphisms. Let $f : D \to E$ be a map of diagrams and let the index sets of $\pi_0(D)$ and $\pi_0(E)$ be A' and B' respectively. We define the index map $\widehat{f} : A' \to B'$ to be the function mapping $(\alpha, i) \in A'$ to the $(\beta, j) \in B'$ such that $f_\alpha\left(D_{\alpha,i}\right) \subseteq E_{\beta,j}$. For all (α, i) such an index (β, j) both exists and is unique because continuous functions map a connected component of the domain to a unique connect of the codomain and f_α is continuous. The definition of the map of diagrams $\pi_0(f)$ can now be succinctly stated; it is the map $\pi_0(f)$ whose component at $D_{\alpha,i}$ is the map

$$\pi_0(f)_{\alpha,i} : D_{\alpha,i} \to E_{\widehat{f}(\alpha,i)}$$

defined as the restriction of f_α to $D_{\alpha,i}$.

This assignment of objects and morphisms clearly preserves compositions and identity morphisms, so π_0 is a functor $Top^{Cat} \to Top^{Cat}$ and we see that it is covariant.

Functoriality of the statistical Mapper

We proceed in a completely analogous fashion with the statistical Mapper, that is, we establish the following composition of functors

$$\mathscr{M}Cov \xrightarrow{\mathfrak{D}} \mathscr{M}^{Cat} \xrightarrow{\mathfrak{C}} \mathscr{M}^{Cat} \xrightarrow{N} sSet \xrightarrow{|-|} Top.$$

Let $\mathscr{M}$ be any of the categories $\mathscr{M}^{\text{gen}}$, $\mathscr{M}^{\text{mon}}$ or $\mathscr{M}^{\text{iso}}$ and let $\mathfrak{C}\colon \mathscr{M} \to \underline{\mathscr{C}}$ be a constant and functorial clustering algorithm.

We define $\mathscr{M}Cov$ to be the following category of covered metric spaces: An object of $\mathscr{M}Cov$ is a pair $(M,\mathscr{U})$ where M is an object of $\mathscr{M}$ and $\mathscr{U}$ is an open cover of M. A morphism $f\colon (M,\mathscr{U}) \to (N,\mathscr{V})$ is an $\mathscr{M}$-morphism such that there for every U of $\mathscr{U}$ is a V of $\mathscr{V}$ satisfying $f(U) \subseteq V$.

We now construct a functor $\mathfrak{D}\colon \mathscr{M}Cov \to \mathscr{M}^{Cat}$ which is completely analogous to $\mathfrak{D}\colon TopCov \to Top^{Cat}$, that is, for an object $(M,\mathscr{U})$ the diagram $\mathfrak{D}(M,\mathscr{U})$ consists of all non-empty intersections U_σ of members of $\mathscr{U}$ and of the corresponding inclusion morphisms. (And, of course, all the metric spaces U_σ of $\mathfrak{D}(M)$ are equipped with the submetric of M.)

In the last subsection we saw how to get a functor π_0 on diagrams of *topological* spaces. In this section we see how to get a functor $\mathfrak{C}$ on diagrams of *metric* spaces.

We now define $\mathfrak{C}$ on $\mathscr{M}^{Cat}$-morphisms: Let $f\colon D \to E$ be a map of diagrams and let the index sets of $\mathfrak{C}(D)$ and $\mathfrak{C}(E)$ be A' and B' respectively. We define the index map $\widehat{f}\colon A' \to B'$ to be the function which maps $(\alpha,i) \in A'$ to the index $(\beta,j) \in B'$ satisfying $f_\alpha(D_{\alpha,i}) \subseteq E_{\beta,j}$. For all (α,i) such an index (β,j) both exists and is unique because $\mathfrak{C}(D_\alpha)$ refines $f_\alpha^{-1}(\mathfrak{C}(E_\alpha))$. The definition of the map of diagrams $\mathfrak{C}(f)$ can now be succinctly stated; it is the map $\mathfrak{C}(f)$ whose component at (α,i) is the map

$$\mathfrak{C}(f)_{\alpha,i} : D_{\alpha,i} \to E_{\widehat{f}(\alpha,i)}$$

defined as the restriction f_α to $D_{\alpha,i}$.

It is easy to see that $\mathfrak{C}$ preserves identities and compositions, so $\mathfrak{C}$ can be regarded as a functor $\mathscr{M}^{Cat} \to \mathscr{M}^{Cat}$ and we see that it is covariant.

In Part III we will have the opportunity to see how one might use the functoriality of Mapper.

9. Relation to homotopy colimits and the nerve theorem

The relation between Mapper and homotopy colimits is well-known, because Gunnar Carlsson mentions the relation briefly in his video lectures at IMA [4], but has not yet been written down and published. My account of this idea stems from Carlsson's video lectures and my advisor Nils Baas.

We argue that Mapper occurs as a natural idea when considering a certain modern proof of the nerve theorem which utilizes homotopy colimits. We therefore give a proof of this theorem. Afterwards we are in a position to dissect the proof and identify one point where we can make a slight improvement. This improvement leads directly to Mapper.

9.1. The nerve theorem and its proof

We need two definitions before stating the theorem.

An open cover $\mathscr{U} = \{U_\alpha\}_{\alpha \in A}$ of a topological space X is said to be a **good cover** if arbitrary intersections of sets of $\mathscr{U}$ are either empty or contractible, that is, for all $S \subseteq A$ we have that $\bigcap_{s \in S} U_s$ is either contractible or empty.

An open cover $\mathscr{U}$ is said to be **numerable** if there exists a partition of unity subordinate to $\mathscr{U}$.

Theorem 9.1.1 (The nerve theorem)**.** *Let X be a topological space and $\mathscr{U}$ an open cover of X. If $\mathscr{U}$ is a good and numerable cover, then X is homotopy equivalent to the nerve of $\mathscr{U}$.*

In our proof of the nerve theorem we prove that both maps in the diagram

$$X \xleftarrow{\pi} \operatorname{hocolim}(X_{\mathscr{U}}) \longrightarrow |N(X_{\mathscr{U}})| \tag{9.1}$$

are homotopy equivalences. This proves the theorem, since the nerves of the cover $\mathscr{U}$ and of the diagram $X_{\mathscr{U}}$ are homotopic, as noted in Section 4.3.

Shifting our attention from the nerve of the cover $\mathscr{U}$ to the nerve of the diagram $X_{\mathscr{U}}$ is key, since this enables the use of homotopy colimits.

The left map

Before defining the left map π of Equation(9.1) we introduce some notation. Let $\mathscr{U} = \{U_\alpha\}_{\alpha\in A}$ be a cover of a space X. For $\sigma \subseteq A$ we denote by $\Delta[\sigma]$ the standard simplex which has the dimension equal to the cardinality of σ, that is, $\Delta[\sigma] = \Delta^{|\sigma|}$. We saw in Section 4.4 that the homotopy colimit of the diagram $X_{\mathscr{U}}$ can be written as a subspace of $X \times \Delta[A]$, namely as the union $\bigcup_{\emptyset\neq\sigma\subseteq A} U_\sigma \times \Delta[\sigma]$. We define π to be the projection onto the first component of $\bigcup_{\emptyset\neq\sigma\subseteq A} U_\sigma \times \Delta[\sigma]$.

By and large, the following proof that the map π is a homotopy equivalence is taken from Segal [15], but the proof had to be adapted to our situation because Segal constructs the space $\operatorname{hocolim}(X_{\mathscr{U}})$ in a different way.

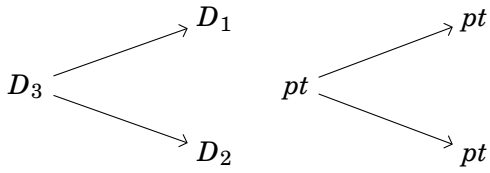
Proposition 9.1.2. *Let X be topological space and let $\mathscr{U}$ be an open cover of X. If $\mathscr{U}$ is numerable, then the canonical projection $\pi\colon \operatorname{hocolim}(X_{\mathscr{U}}) \to X$ is a homotopy equivalence.*

Proof. Denote by π the projection $\operatorname{hocolim}(X_{\mathscr{U}}) \to X$. In this proof we construct a map $\Psi : X \to \operatorname{hocolim}(X_{\mathscr{U}})$ with $\pi \circ \Psi = \mathrm{id}_X$ which embeds X as a (strict) homotopy retract of $\operatorname{hocolim}(X_{\mathscr{U}})$. This proves the proposition.

Fix an open cover $\mathscr{U} = \{U_\alpha\}_{\alpha\in A}$ and a partition of unity $\{\psi_\alpha : U_\alpha \to [0,1]\}_{\alpha\in A}$ subordinate to $\mathscr{U}$. We define the function $\Psi : X \to \operatorname{hocolim}(X_{\mathscr{U}})$ to be $\mathrm{id}_X \times \psi$ with $\psi : X \to \Delta[A]$ a linear combination of vertices of $\Delta[A]$

$$\psi(x) = \sum \psi_\alpha(x)\alpha,$$

where the sum ranges over those $\alpha \in A$ so that x is contained in U_α. Note that we use the symbol α to denote both the index of an open set U_α and the corresponding vertex of $\Delta[A]$. This map Ψ is well-defined because $\operatorname{hocolim}(X_{\mathscr{U}})$ is the space $\bigcup_{\emptyset\neq S\subseteq A} X[S] \times \Delta[S]$ and a point in the standard simplex $\Delta[S]$ is uniquely specified by a linear combination of the vertices α of S where the coefficients sum to 1. The map Ψ is also a homeomorphism onto its image; a regular argument involving the partition of unity shows continuity and a continuous inverse is given by the projection π restricted to $\Psi(X)$.



(a) The diagram D (b) The diagram D^{pt}

Figure 9.1.: An illustration of the action of pt on diagrams. We see the diagram D and the diagram D^{pt}

The subset $\Psi(X) \subseteq \operatorname{hocolim}(X_{\mathscr{U}})$ is a strict deformation retract of $\operatorname{hocolim}(X_{\mathscr{U}})$. Indeed, the straight line homotopy $F : \operatorname{hocolim}(X_{\mathscr{U}}) \times [0,1] \to \Psi(X)$ given by

$$F((x,u),t) = \big(x,(1-t)u + t\psi(x)\big)$$

is such a retract. □

The right map

We now define the right map of Equation (9.1).

Let $\mathscr{U} = \{U_\alpha\}_{\alpha \in A}$ be an open cover of a space X. Recall that the nodes of the diagram $X_{\mathscr{U}}$ are the intersections U_σ for subsets σ of A. Let $(X_{\mathscr{U}})^{pt}$ be the diagram with the same indexing set as $X_{\mathscr{U}}$, but with each node replaced by a one-point space pt, see Figure 9.1. We define the map of diagrams $f : X_{\mathscr{U}} \to (X_{\mathscr{U}})^{pt}$ to be the natural transformation with components $f_\sigma : U_\sigma \to pt_\sigma$. This shows that f is a map of diagrams because the needed commutativity of maps is trivially satisfied.

The space $|N(X_{\mathscr{U}})|$ is the space $\operatorname{hocolim}\big((X_{\mathscr{U}})^{pt}\big)$ by our definition of the nerve, so we can define the right map of (9.1) to be the map

$$\operatorname{hocolim}(f) : \operatorname{hocolim}(X_{\mathscr{U}}) \to \operatorname{hocolim}(X_{\mathscr{U}})^{pt}.$$

The next proposition states that $\operatorname{hocolim}(f)$ is a homotopy equivalence.

Proposition 9.1.3. *Let $\mathscr{U}$ be a cover of a topological space X. If $\mathscr{U}$ is a good cover, then the map* $\operatorname{hocolim}(f) : \operatorname{hocolim}(X_{\mathscr{U}}) \to \operatorname{hocolim}\big((X_{\mathscr{U}})^{pt}\big)$ *is a homotopy equivalence.*

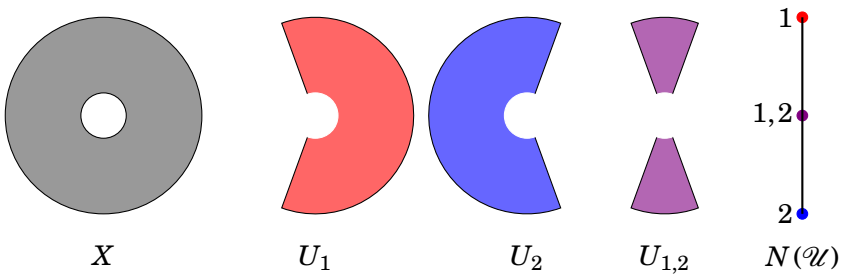


Figure 9.2.: The topological space X, a cover $\mathcal{U} = \{U_1, U_2\}$ of X, and the nerve $N(X_{\mathcal{U}})$ of $\mathcal{U}$. The homotopy type of $N(\mathcal{U})$ and X is different, as is allowed by the nerve theorem since the intersection $U_{1,2}$ has two connected components and is therefore not contractible.

Proof. All the nodes of the diagram $X_{\mathcal{U}}$ are contractible, because $\mathcal{U}$ is a good cover. Therefore, all the components $f_\sigma : U_\sigma \to pt$ of f are homotopy equivalences. Each of the components of f is then a homotopy equivalence, so the induced map $\mathrm{hocolim}(f)$ a homotopy equivalence, due to Proposition 3.4.1. □

This finishes the proof of the nerve theorem, because Proposition 9.1.3 and Proposition 9.1.2 together with transitivity of homotopy equivalences give the desired homotopy equivalence $X \simeq |N(\mathcal{U})|$.

9.2. The "Mapper" theorem and its proof

We present two examples of covers and their nerves. In both examples the covers are not good and the nerves do not preserve the homotopy type of the space they cover. In the first case we are able to fix the difference in homotopy type by a slight change to the nerve construction and this change leads us to Mapper and an analogue to the nerve theorem, Theorem 9.2.2, which holds for Mapper.

Example 9.2.1. Consider Figure 9.2 showing a space X and an open cover $\mathcal{U}$ of X which does *not* satisfy the assumptions of the nerve theorem, because the intersection $U_{1,2}$ has too many connected components. □

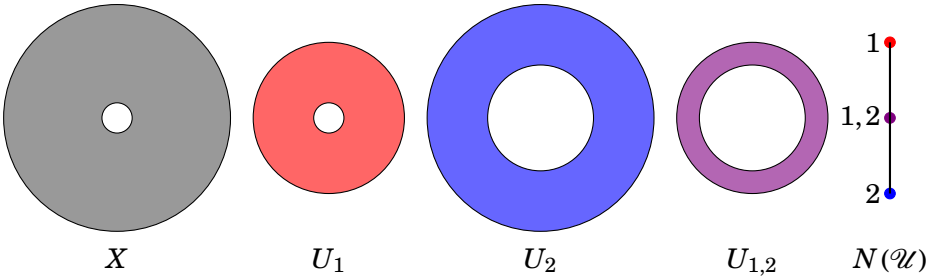
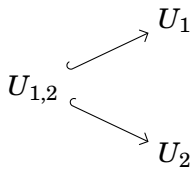


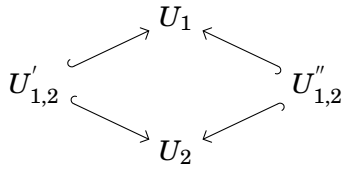
Figure 9.3.: The topological space X, a cover $\mathscr{U} = \{U_1, U_2\}$ of X, and the nerve $N(X_{\mathscr{U}})$. The homotopy type of $N(\mathscr{U})$ and X is different, as is allowed by the nerve theorem since the intersection $U_{1,2}$ has non-trivial fundamental group and is therefore not contractible.

Example 9.2.2. Consider Figure 9.3 showing a space X and an open cover $\mathscr{U}$ which does *not* satisfy the assumptions of the nerve theorem because the intersection $U_{1,2}$ has non-trivial fundamental group. □

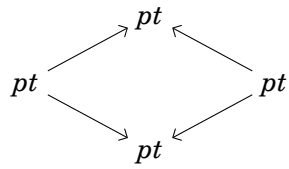
We study the first example in some detail. The diagram $X_{\mathscr{U}}$ has the form



and we see that, because the intersection $U_{1,2}$ has two connected components, Proposition 9.1.3 does not guarantee that $\operatorname{hocolim}(X_{\mathscr{U}}) \to \operatorname{hocolim}\left((X_{\mathscr{U}})^{pt}\right)$ is a homotopy equivalence. We therefore construct a diagram in which all components *are* contractible and see what conclusions we can draw. Denote by $U'_{1,2}$ and $U''_{1,2}$ the two connected components of $U_{1,2}$. The diagram $\pi_0(X_{\mathscr{U}})$ has the form



and all spaces in this diagram are contractible. Therefore, the diagram $\pi_0(X_{\mathscr{U}})$ is component-wise homotopy equivalent to the diagram



which is the diagram $\pi_0(X_{\mathscr{U}})^{pt}$.

We summarize this discussion in the following proposition, the proof of which follows directly from the fundamental property of homotopy colimits, Proposition 3.4.1.

We call a collection of open subsets $\mathscr{U} = \{U_\alpha\}_{\alpha \in A}$ of a space X is π_0**-good** if for all $\sigma \subseteq A$ the intersection U_σ is either empty or has contractible connected components. All good covers are π_0-good, but not vice versa.

Proposition 9.2.1. *Let $\mathscr{U}$ be an open cover of a topological space X. If the cover $\mathscr{U}$ is π_0-good, then we have the following homotopy equivalence*

$$\operatorname{hocolim}(\pi_0(X_{\mathscr{U}})) \simeq \operatorname{hocolim}\left(\pi_0(X_{\mathscr{U}})^{pt}\right).$$

□

Note. The space $\operatorname{hocolim}\left(\pi_0(X_{\mathscr{U}})^{pt}\right)$ is $|N(\pi_0(X_{\mathscr{U}}))|$ which we recognize as Mapper of the cover $\mathscr{U}$. □

Inspired by the previous discussions, we now aim at proving the following "Mapper theorem", an analogy to the previous "nerve theorem".

Theorem 9.2.2. *Let $\mathscr{U}$ be an open cover of a topological space X. If the cover $\mathscr{U}$ is both numerable and π_0-good, then the space X is homotopy equivalent to Mapper$(X_{\mathscr{U}})$.*

The proof proceeds by showing that each of the maps of the diagram

$$X \leftarrow \operatorname{hocolim}(X_{\mathscr{U}}) \leftarrow \operatorname{hocolim}(\pi_0(X_{\mathscr{U}})) \rightarrow \operatorname{hocolim}\left(\pi_0(X_{\mathscr{U}})^{pt}\right)$$

are homotopy equivalences. The left map is the map of Proposition 9.1.2 and the right map is the map of Proposition 9.2.1, so the only map we need to discuss is the middle map, which we now define.

Let the index category of the diagram $X_{\mathscr{U}}$ be A and the index category of $\pi_0(X_{\mathscr{U}})$ be A'. We recall our usual notation, we have spaces U_σ in $X_{\mathscr{U}}$ and spaces $U_{\sigma,a}$ in $\pi_0(X_{\mathscr{U}})$ $U_{\sigma,a}$ is a connected component of U_σ. Here, σ is an object of A and (σ,a) is an object of A'.

We define the map of diagrams

$$\theta\colon \pi_0(X_{\mathscr{U}}) \to X_{\mathscr{U}} \tag{9.2}$$

to be the map whose component at (σ,a) is the inclusion $U_{\sigma,a} \hookrightarrow U_\sigma$. It is easily verified that this is a well-defined map of diagrams.

Proposition 9.2.3. *Let $\mathscr{U}$ be an open cover of a topological space X, and let $\theta\colon \pi_0(X_{\mathscr{U}}) \to X_{\mathscr{U}}$ be the map defined in Equation* (9.2) *above. The map*

$$\operatorname{hocolim}(\theta)\colon \operatorname{hocolim}(\pi_0(X_{\mathscr{U}})) \to \operatorname{hocolim}(X_{\mathscr{U}})$$

is a homotopy equivalence.

Note. The proof of this proposition is not particularly enlightening and we urge the reader to draw the spaces $\operatorname{hocolim}(\pi_0(X_{\mathscr{U}}))$ and $\operatorname{hocolim}(X_{\mathscr{U}})$ for some spaces X and covers $\mathscr{U}$, say the ones of Figure 9.2 and Figure 9.3. One might then come to the same conclusion as the author, that the proposition is not deep, and that the only challenge with the proof is keeping track of combinatorics and notation. This is a challenge because we in this proof have to really dig into the definition of the homotopy colimit — we can not rely on the property of Proposition 3.4.1 as we have done before. □

Proof. We denote by U_{σ_i} spaces in the diagram $X_{\mathscr{U}}$. We denote the connected components of U_{σ_i} by $U_{\sigma_i,a}$ for $a \in A_{\sigma_i}$ where A_{σ_i} is just an index set. All the maps of the diagram $X_{\mathscr{U}}$ are inclusions and we denote the inclusion $U_{\sigma_i} \to U_{\sigma_j}$ by f_{σ_i}.

We write out the definition of both $\operatorname{hocolim}(\pi_0(X_{\mathscr{U}}))$ and $\operatorname{hocolim}(X_{\mathscr{U}})$:

$$\operatorname{hocolim}(X_{\mathscr{U}}) = \left[\coprod_n \left(\coprod_{f_{\sigma_0}\to\cdots\to f_{\sigma_n}} U_{\sigma_0}\right) \times \Delta^n\right] \Big/ \sim,$$

$$\operatorname{hocolim}(\pi_0(X_{\mathscr{U}})) = \left[\coprod_n \left(\coprod_{f_{\sigma_0,a_0}\to\cdots\to f_{\sigma_n,a_n}} U_{\sigma_0,a_0}\right) \times \Delta^n\right] \Big/ \sim.$$

The following series of homeomorphisms show that the two spaces are homeomorphic before we pass to quotient spaces.

$$\begin{aligned}
\coprod_n \left[\left(\coprod_{f_{\sigma_0} \to \ldots \to f_{\sigma_n}} U_{\sigma_0} \right) \times \Delta^n \right] &= \coprod_n \left[\left(\coprod_{f_{\sigma_0} \to \ldots \to f_{\sigma_n}} \left(\coprod_{a \in A'_{\sigma_0}} U_{\sigma_0, a} \right) \right) \times \Delta^n \right] \\
&= \coprod_n \left[\left(\coprod_{f_{\sigma_0} \to \ldots \to f_{\sigma_n}} \left(\coprod_{a \in A'_{\sigma_0}} U_{\sigma_0, a} \right) \times \Delta^n \right) \right] \\
&= \coprod_n \left[\coprod_{f_{\sigma_0} \to \ldots \to f_{\sigma_n}} \left(\coprod_{a \in A'_{\sigma_0}} U_{\sigma_0, a} \times \Delta^n \right) \right] \\
&= \coprod_n \left[\coprod_{f_{\sigma_{0,a}} \to \ldots \to f_{\sigma_{n,a_n}}} \left(U_{\sigma_0, a} \times \Delta^n \right) \right].
\end{aligned}$$

The fact that the quotient spaces are homeomorphic follows from the observation that the points (x, s) and (y, t) in $\operatorname{hocolim}(\pi_0(X_{\mathscr{U}}))$ are equivalent if and only if the points $(\theta(x), s)$ and $(\theta(y), t)$ in $\operatorname{hocolim}(X_{\mathscr{U}})$ are equivalent. We now show the forward implication to be true. Let x be a point in $U_{\sigma_i, a}$ and y a point in $U_{\sigma_j, b}$. By assumption, (x, s) and (y, t) are equivalent, so there is a map φ of the simplex category Δ such that $\varphi_*(x)$ equals y and $\varphi^*(t)$ equals s. (Here, φ_* is the map $srep\,(\pi_0(X_{\mathscr{U}}))(\varphi)$.) This same map φ shows that $(\theta(x), s)$ and $(\theta(y), t)$ are equal since by commutativity $\varphi_*(\theta(x))$ equals $\theta(y)$ and $\varphi^*(t)$ equals s as before. (Here, φ_* is the map $srep\,(X_{\mathscr{U}})(\varphi)$.) A similar argument works for the backward implication. □

This completes the proof of the "Mapper" theorem, Theorem 9.2.2.

Afterthoughts

The previous result concerns the topological Mapper only, and we should ask ourselves what significance the work has for our purpose of studying point clouds, so we elaborate on this in the next few paragraphs.

We take the point of view that we have, somehow, gotten hold of a cover $\mathscr{U}$ of a topological space X and that we want to make the best possible study of X using only a computer. We know that $\operatorname{hocolim}(X_{\mathscr{U}})$ (or even the union of members of $\mathscr{U}$) recovers the homotopy type of the space X, but it is

hopelessly difficult to encode the complete topology of a space in a computer, so we see ourselves forced to reduce the amount of information to a modest amount of combinatorial data. The normal, or at least the most traditional, way to do this is to take the nerve of $\mathscr{U}$. This recovers the homotopy type of X if $\mathscr{U}$ is good. In Theorem 9.2.2 we have seen that there is a better way of reducing X to combinatorial data, namely by taking Mapper of $\mathscr{U}$. This is better since we recover the homotopy type of X even when $\mathscr{U}$ is only π_0-good. We therefore think of Mapper as being a type of nerve construction, but with one improvement, namely that Mapper comes with a zeroth-order fix of homotopy type included.

The situation is not quite as easy in applications, because there our cover $\mathscr{U}$ is of a finite metric space M. We imagine that the points of M are sampled from some topological space X and that our mission is to approximate the homotopy type of X using only the cover $\mathscr{U}$. We have no theorem stating whether $Mapper(\mathscr{U})$ or $N(\mathscr{U})$ is most likely to recover the homotopy type of X, but we have some faith in that the results from the topological Mapper have relevance also for the statistical situation. We will see if this is the case in the numerical experiments of Part III.

Part III.

Experiments

10. The implementation of Mapper

There is an implementation of Mapper available at [16] written by the computational topology group at Stanford. Unfortunately, ever since they founded the firm "Ayasdi" this code has not been developed. The code [16] is still in a buggy state and is unnecessarily slow, so we therefore modified it somewhat. The difference in runtime between their Mapper implementation and ours grew with the size of the point cloud; the most extreme example is the calculation of an example which we did not have time to include into this thesis, which with their implementation took six hours and with ours only two seconds. Both of these calculations was done on the same machine. We suspect that the reason for this discrepancy is that they for their computations used Matlab's class for sparse matrices while we used the standard and more memory demanding class of matrices.

The code

Our implementation of Mapper is written in the commercial closed source mathematics software *Matlab*. We chose Matlab because the original implementation [16] is written in Matlab, and therefore we could reuse some of its routines, especially the output routines,

The clustering method we use in our implementation is single-linkage clustering with a fixed prune value. The method we use to find this prune value is the same as used in the original implementation and is described in [17]. We wanted to write an implementation where one can change the clustering algorithm between each run, but, because the interfaces to Matlab's clustering-routines vary, it would take too much time to write. However, with an ad-hoc method, we have been able to run our implementation of Mapper with k-means clustering as well, see Section 11.2.

The computational part of the program consists only of bookkeeping and

there are no algorithmic difficulties to speak of. Therefore, we choose to make no further comments on the code.

Input and initialization

The input to our program is a finite metric space, a filter function, and two parameters n and p. The program creates as parameter space the interval $[a,b]$, where a and b are the maximum and minimum of the filter function on the metric space, and creates the cover $\mathscr{U}(n,p)$ as explained in Section 7.3. The program calculates Mapper with respect to this cover.

Output

Our implementation of Mapper outputs a *dot-file*. A dot-file is a text-file encoding the nodes and edges of a graph. The file can be read by the open-source software suite *Graphviz* (Graph Visualization Software) [1] and Graphviz processes this file and outputs a nicely typeset graph, as seen in Figure 10.1. The graph encoded in the dot-file is the 1-skeleton of the simplicial complex that Mapper outputs. We therefore loose all information encoded in n-cells for $n \geq 2$. The reason we do this is simplicity, Graphviz is a simple solution to plotting graphs, and there exists no equivalent solution plotting simplicial complexes, or even 2-skeletons of simplicial complexes. We could have written such a plotting routine ourself, but in order to do this we would have to meddle with 3D-programming and this goes beyond the scope of this Master's thesis. Also, the simplicity of a graph makes the plots easier to read.

The dot-file created by our Mapper implementation specifies not only the 1-skeleton Γ, but it also specifies a certain size and color of each node N of Γ, as is explained in Figure 10.1. Let U_σ be the set corresponding to the node N. The size of N represents the number of points in U_σ. The color of N represents the mean value of the filter function on U_σ, blue means a low filter value and red means high. With this information we can read off even more information directly from the output graph.

Lastly, we can label each node with some text. What we choose this label to represent will vary from application to application, so we will specify what it means when we need it.

Performance

It is hard to give the computational complexity of the Mapper algorithm. The runtime depends on the size of the point cloud, the number of bits needed to specify a point, the number of spaces in the diagram $X_{\mathscr{U}}$, and the complexity of both the clustering algorithm and the filter function. It is especially hard to analyze how many spaces there will be in the diagram $X_{\mathscr{U}}$, because we can construct examples where adding only one point to a point cloud might more than double the number of spaces. Thus, a deeper analysis is needed in order to give a good account of Mapper's runtime, but we try to give a feel for the runtime with the following unscientific anecdotes.

All our computations were practically instant, with the longest computation finishing in less than two seconds.

When comparing the runtime of Mapper to the runtime of single-linkage clustering itself, we find that for most choices of parameter values it is faster to run Mapper. When increasing the parameter p of $U(n,p)$, that is, when increasing the overlap percentage of the cover, the diagram $X_{\mathscr{U}}$ grows and the Mapper computation takes more time. At some choice of p single-linkage clustering and Mapper are equally time-consuming; in our experiments this happened at $p \approx 90\%$.

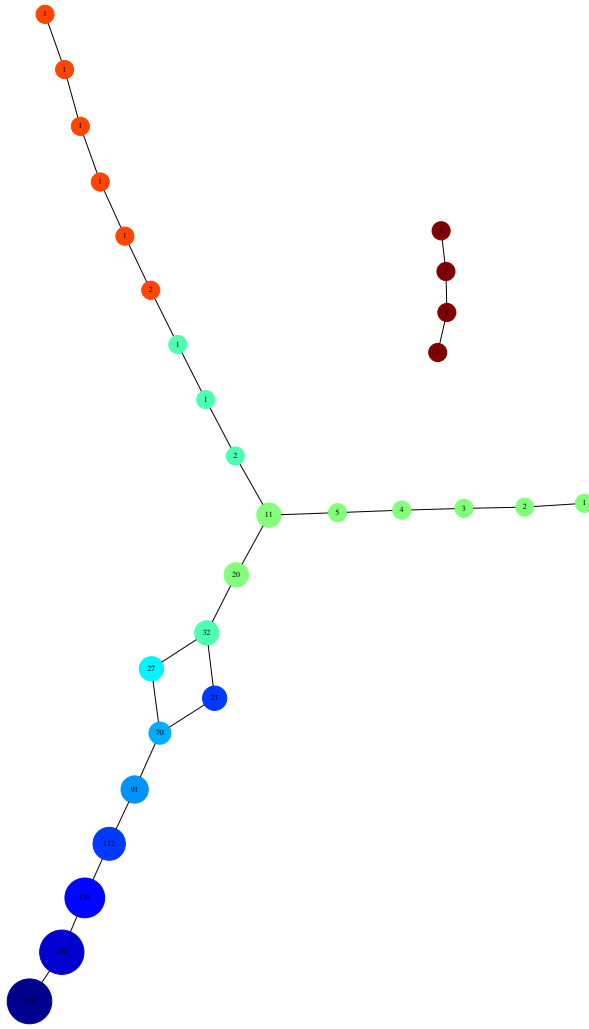


Figure 10.1.: Example of output from our Mapper implementation. The graph is the 1-skeleton of the simplicial complex output from Mapper. The size of a node represents the number of points in that node and the color represents the mean value of the filter function on that node.

11. The Fisher-Anderson iris flower data set

To present our methods we want a dataset with a known taxonomy which is easy to visualize while still being non-trivial to cluster. That is, the dataset should be low-dimensional and the data set's natural taxonomy should not correspond neatly to different clusters. For this purpose we have chosen the classic *Fisher-Anderson iris dataset* [9].

Anderson was a botanist which in the mid-thirties studied three species of iris flowers. He gathered 150 flowers, 50 of each species, and measured the length and width of both petals and sepals. Fisher, the famous statistician and biologist, used this dataset to showcase his newfound "linear discriminant model"; he wanted to identify the species of a flower solely on the basis of Anderson's measurements. Of the three species — *iris setosa*, *iris versicolor* and *iris virginica* — iris setosa was easy to distinguish, but versicolor and virginica was not, even with Fisher's linear discriminants. Since then, the dataset has become a classic testbed for taxonomic procedures like, for instance, clustering.

Let us first get a feel for how the data looks. Since the dataset is four-dimensional we can not plot it directly, so we choose to plot projections of the data onto two dimensional planes, as seen in Figure 11.1. In this figure, we see that the iris setosa (in red) are clearly separated from the other two species, but the iris virginica (blue) and versicolor (green) are somewhat intermingled in all of the 12 projections. We can not conclude, however, that the virginicas and the versicolors are indistinguishable, as the intermingling of points may be an artifact of the linear projections. What we *do* conclude is that at most only a few iris versicolor and virginicas overlap, and that most of the blue and green points lie in quite different regions of the space. It is therefore unclear if our clustering algorithms will be able to separate the species or not, but the Mapper algorithm might be able to reveal this kind of structure in the data.

11.1. Cluster analysis

We have run all our clustering algorithms on this dataset and plotted the results in Figure 11.2. The plane shown in the figure is petal length by petal width, and points of different clusters are marked with different colors and shapes. In the following paragraphs we comment on the performance of the clustering algorithms and on what conclusions we can draw based on the different read-outs. All clustering algorithms managed to partition the iris setosa correctly, so we will comment only on their ability to distinguish versicolors from virginicas.

k-means clustering. Consider Figure 11.2a where the result of the k-means algorithm is plotted. We ran the k-means algorithm with k set to 3 because we know that this is the correct number of clusters. It should be noted that in most situations we do not know how to choose k and that we therefore have to set k to some arbitrary value, and that this arbitrary choice greatly affects the final partitioning. The part of the point cloud containing versicolors and virginicas is cut in two at approximately the right spot, causing most points to be partitioned correctly. One might wonder if this is due to an gap between the versicolors and virginicas or if it happened due to some artifact of the algorithm. For example, the k-means algorithm tends to produce approximately (hyper)spherical clusters [11] and it might be this tendency which forced the cloud to be partitioned more or less correctly, even if no gap between versicolor and virginicas exists.

Dbscan clustering. We ran the Dbscan algorithm with $m = 5$ and $\varepsilon = 0.5$ and the result is shown in Figure 11.2b. We chose these parameters with the automatic procedure proposed in the original article [8] and they worked quite well in that they produced a reasonable ratio of big clusters to one-point clusters. However, the Dbscan algorithm does not distinguish between virginicas and versicolors at all. And further, even by fine-tuning the parameter values we could not get Dbscan to distinguish between versicolors and virginicas. This suggests that there is no space between the versicolors and virginicas which are “less dense”, because then the Dbscan algorithm would have partitioned them correctly.

Note. In the Dbscan plot in Figure 11.2b it looks like some of the one-point

clusters produced by Dbscan lie almost in the middle of another cluster; this is an artifact of the projection chosen and it does not show up in many of the other projections. □

Single-linkage clustering We cut the single-linkage dendrogram such that we obtain three clusters and this clustering is shown in Figure 11.2c.

The single-linkage clustering does not distinguish between the versicolors and virginicas at all. Furthermore, when we cut the single-linkage dendrogram to get more clusters than three, only one-point and two-point clusters appear. Thus, as single-linkage clustering clusters only according to distance, we conclude that the smallest distance between virginicas and versicolors is smaller than the smallest distances within each cluster.

Complete and average linkage We ran both average linkage and complete linkage clustering on the point cloud, but the average linkage output is not shown since it performed almost exactly like complete linkage clustering. The complete linkage clustering corresponds quite well to the true taxonomy, but, as with k-means clustering, this is due to us knowing the exact number of clusters, which we in general do not. In Figure 11.3 the complete linkage output with 5 clusters is shown and we see that it produces clusters of reasonable size which are quite compact, as is typical of the method [11]. This feature makes it hard to *a priori* decide what the correct number k of clusters is, as many choices of k give reasonable looking clusters.

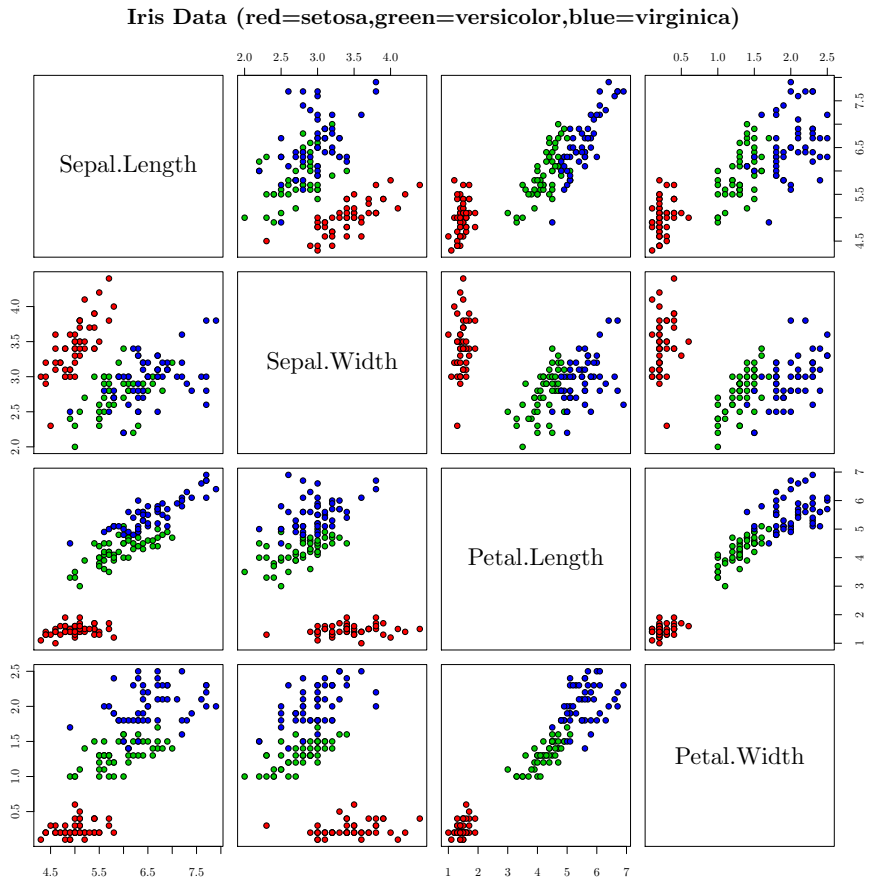
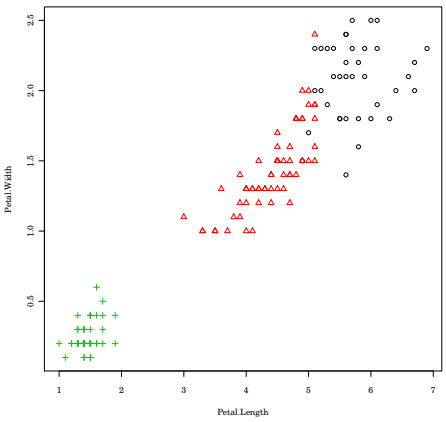
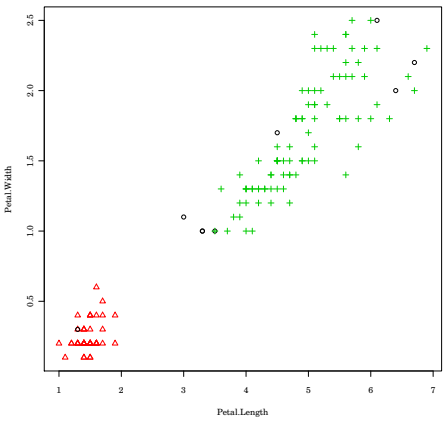


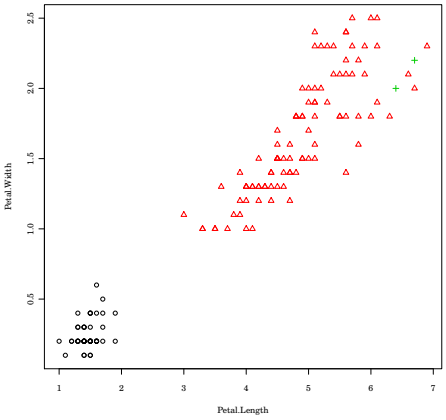
Figure 11.1.: The Fisher-Anderson iris dataset is a four-dimensional dataset consisting of 150 points and each point is determined by four measurements of an iris flower, petal and sepal width and length. In the figure the projections onto all the different planes spanned by pairs of standard basis vectors is shown. Three different species of iris flowers correspond to the different colors in the plots.



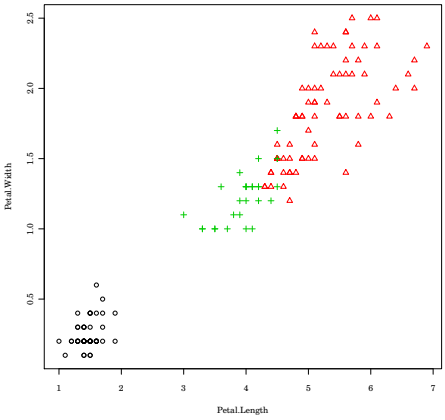
(a) 3-means



(b) Dbscan, circles denote one-point clusters



(c) Single linkage



(d) Complete linkage

Figure 11.2.: Four 2D plots of petal length and petal width of the iris data. The different plots correspond to different clustering algorithms. The points which lie in different clusters are distinguished by their color and shape.

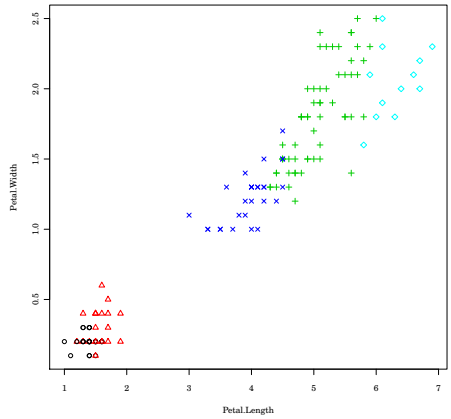


Figure 11.3.: Complete linkage clustering of the Iris data with the number of clusters set to 5.

11.2. Mapper analysis

To kick off the presentation of the Mapper analysis, we present an example where we have tweaked the parameter values until we found an output we liked. Afterwards, after we have learned how to read the Mapper-output, we show how we might go about choosing or finding the parameter values in situations where we do not know the answers beforehand.

Analysis with good choice of parameter values

We start with the analysis where we have cheated by tweaking the parameter values until we find parameters which fit our data. We mean by cheating that we use information that we would not have available in a normal use-case. Note that we also cheated during the cluster analysis, by specifying the correct amount of clusters for some of the clustering algorithms.

Initialization

Let X be the Iris data set regarded as a metric space with the Euclidean metric. We found that the following choices of filter and parameter space gave satisfying results: We chose as our filter the function $f : X \to \mathbb{R}$ given by $f(x) = ecc_1(x) \cdot density_4(x)$, and the range of this filter on X is 0.18 to 0.35 so we define the parameter space to be $[a,b] = [0.18, 0.35] \subset \mathbb{R}$. We cover the interval $[a,b]$ with $n = 9$ intervals $U_1, \ldots, U_9$, each of length $l = \frac{3}{10}(b-a)$. Consecutive U_i overlap $p = 60\%$, and this means that three consecutive intervals; U_i, U_{i+1} and U_{i+2}; have non-empty intersections. Therefore we potentially have 2-simplices in the Mapper output. There will be no 3-simplices, since every intersection of four U_i will be empty.

Reading the Mapper output

Consider Figure 11.4, where we have plotted the graph $\Gamma = Mapper\left(f^{-1}(\mathscr{U})\right)$. If $Mapper\left(f^{-1}(\mathscr{U})\right)$ contains any 2-simplices, then Γ will only its 1-skeleton, because of the mentioned troubles with plotting higher simplices. The text in each node tells us how many versicolors, virginicas and setosas which are contained in that node.

We can extract a clustering of the point cloud by calculating the simplicial homology $H_0(\Gamma)$. We see that the figure contains two connected components,

one containing only setosas and one containing only versicolors and virginicas. Thus, if reduced to a clustering algorithm, Mapper with these choices of parameters gives the same result as single-linkage clustering.

We now see what information can be extracted from the graph's geometry and from the nodes' size and color.

The connected component of Γ containing only setosas, let us call it Γ_S, consists of seven nodes. The majority of the flowers lie in the four big nodes, while the node to the far left and the two blue nodes to the right might be considered outliers. The range of filter values on Γ_S range from very high (deep red color) to the quite low (medium blue color). The red nodes contain eccentric dense points and the blue nodes contain non-eccentric sparse points. The wide range of colors observed in Γ_S shows that the iris setosa are very heterogeneous, although most of the flowers belong in the middle nodes which vary little in color and are therefore homogeneous.

The connected component of Γ containing only versicolors and virginicas, let us call it Γ_V, also consists of seven nodes, but the nodes are on average bigger, so there are more points belonging to Γ_V than to Γ_S. The majority of the flowers lie in the five rightmost nodes, but only the node to the far left can be considered an outlier. The rightmost node contains only versicolors, the two leftmost nodes contain only virginicas, and the nodes in-between varies gradually between these extremes, with the node in the middle containing approximately as many versicolors as virginicas. The range of filter values on Γ_V range from medium high (orange) to very low (dark blue), and it looks like the filter function f has a lower range over Γ_V than over Γ_S, possibly implying that it is less variation between virginicas and versicolors than there is between setosas.

Choosing parameter values

We present two figures, Figure 11.5 and Figure 11.6, where the Mapper output of the iris data for many different choices of parameters are shown, and discuss how we on the basis of these figures can decide if a particular choice of parameter values are good or not. We will argue that functoriality is the key ingredient for making parameter choices in this way.

For $l < 1$ let $\mathscr{U}(l,p)$ be the cover of of the interval $(b-a)$ which consists of intervals of length $l(b-a)$ and overlap percentage p, as explained in Section 7.3. This means that a value of $l = 0.3$ gives a cover of $(b-a)$ which

consists of intervals of length $0.3(b-a)$. Denote by $\Gamma(l,p)$ the graph that our implementation of Mapper outputs when used on the cover $\mathscr{U}(l,p)$.

Mapper output for $l = 0.3$

In Figure 11.5 the output of Mapper used on the iris data for a fixed value of $l = 0.3(b-a)$ and for many values of p. In Figure 11.5a the overlap percentage p is set to 1%, which means that the cover is almost non-overlapping. Still, two big clusters are formed, but there are also three smaller clusters. With $p = 4\%$ in Figure 11.5b the smaller clusters have merged into the two bigger clusters, and we begin to see the basic geometry which is stable under all the other choices of p, namely that the clusters form two strands. At $p = 75\%$ one of the strands has grown two "horns", and this feature persists and becomes clearer with higher values of p.

We have taken care to choose values of l and p such that the cover $\mathscr{U}(l,p)$ refines the next, that is, $\mathscr{U}(0.3,1\%) \to \mathscr{U}(0.3,4\%) \to \mathscr{U}(0.3,18\%) \to \mathscr{U}(0.3,30\%) \to \mathscr{U}(0.3,90\%)$. This means that we have the chain of refinements

$$f^{-1}(\mathscr{U}(0.3,1\%)) \to f^{-1}(\mathscr{U}(0.3,4\%)) \to \cdots \to f^{-1}(\mathscr{U}(0.3,90\%)),$$

and, by functoriality, we have the induced simplicial maps

$$\Gamma(0.3,1\%) \to \Gamma(0.3,4\%) \to \cdots \to \Gamma(0.3,90\%).$$

Our implementation of Mapper does not have the capability to calculate these induced maps, but we can still draw some conclusions from them. The following argument will be used repeatedly in the following: The induced maps are continuous, so each connected component of the domain are mapped to a unique component of the codomain. Therefore, as p grows, no connected component of $\Gamma(0.3,p)$ can "split" to form two components. We see in Figure 11.5 that this holds true.

Mapper output for $l = 0.1$

In Figure 11.6 the output of Mapper used on the iris-data for a fixed value of $l = 0.1$ is shown. We expect these graphs to be more disconnected than for $l = 0.3$ because a smaller value of l makes the cover of the parameter

space less overlapping. We also expect these graphs to have more nodes in them, because a lower value of l leads to more covering sets. Both of these expectations hold true, there are more connected components in each of the subfigures of Figure 11.6 and also more nodes. Although the pictures for $l = 0.1$ and $l = 0.3$ are quite different for low values of p we see that for $p = 90\%$ they are more or less equal, but with one more connected component for $l = 0.3$.

The argument above, that connected components can not split as p grows, holds true also for $l = 0.1$ because also for this value of l we have the refinements

$$\mathscr{U}(l, 1\%) \to \mathscr{U}(l, 4\%) \to \cdots \to \mathscr{U}(l, 90\%).$$

Using *persistence* to choose parameter values

We come to very different conclusions about the nature of our point cloud depending on which of the Mapper-outputs we look at. For instance, from $\Gamma(0.10, 1\%)$ in Figure 11.6a we might think that the Iris point cloud is very disconnected. However, since we in the above saw that we from our notion of functorial clustering got induced maps between $\Gamma(0.1, p)$ for $p = 1\%, 4\%, 18\%, 30\%, 75\%$ and 90% we can use the idea of persistence to decide on proper values for the parameters l and p.

In Figure 11.5a we see that there in $\Gamma(0.30, 1\%)$ are five connected components, but that these quickly disappear already in $\Gamma(0.30, 4\%)$, and we therefore have little belief in that they represent important features of the point cloud. However, the two bigger clusters persist through all the values of p, so we feel confident that they represent an important feature of the Iris point cloud.

In Figure 11.6 the same general phenomena are present, but the smaller clusters are more persistent than in Figure 11.5. The most persistent of these clusters is the deep red one. By examination of the points which are part of this deep red component we see that it consists of the point which we in Figure 11.4 speculated was an outlier. That this component persists over many values of p strengthens this belief.

After these examinations we believe that no single Mapper output gives us a completely correct picture of our data, but that by examination of one or more series $\Gamma_0 \to \Gamma_1 \ldots \to \Gamma_n$ of Mapper outputs we can get a good impression of the point cloud.

Mapper with a non-functorial clustering algorithm

We noted in the introduction, that in our implementation of Mapper one can not change the clustering algorithm easily. However, with lots of manual labor we have managed to run some examples with k-means clustering. We now present these results. The point of these explorations is to see what happens when we run Mapper with a non-functorial clustering algorithm.

We ran the 3-means clustering with the parameters of the covering space being $l = 0.3$ and the p values the same as before. The result is shown in Figure 11.7.

We want to highlight the figures Figure 11.7d and Figure 11.7e which show the two graphs $\Gamma(0.30, 30\%)$ and $\Gamma(0.30, 75\%)$. Since $\mathscr{U}(0.30, 30\%)$ refines $\mathscr{U}(0.30, 75\%)$ we would with a functorial clustering algorithm have an induced map $\Gamma(0.39, 30\%) \to \Gamma(0.30, 75\%)$ which prohibits the components of $\Gamma(0.30, 30\%)$ to split and form two components in $\Gamma(0.30, 75\%)$. However, this is exactly what happens when we use 3-means clustering.

Lastly, there are no features of the point cloud which clearly persists through many parameter values. On the basis of the results in this subsection we can not even conclude that there are two clusters in the point cloud.

We believe that we get such inconclusive results because 3-means clustering is not functorial.

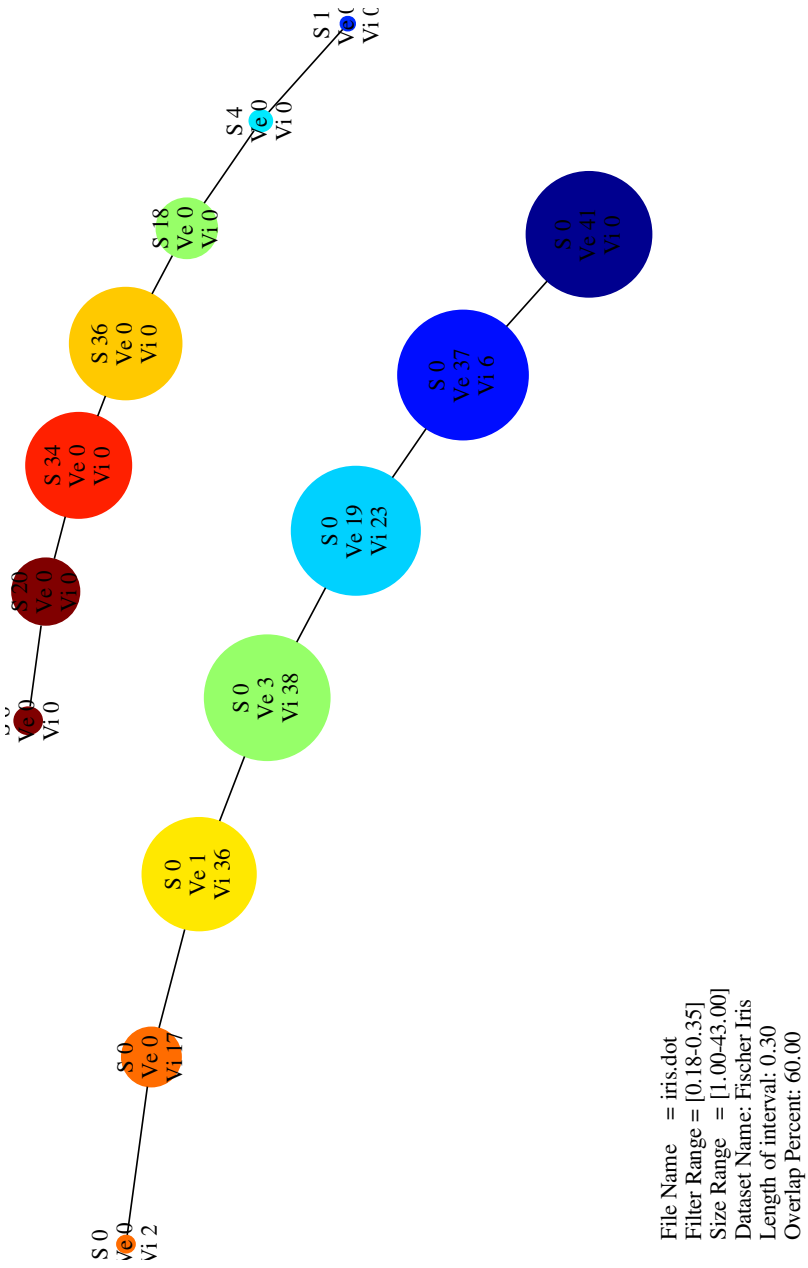
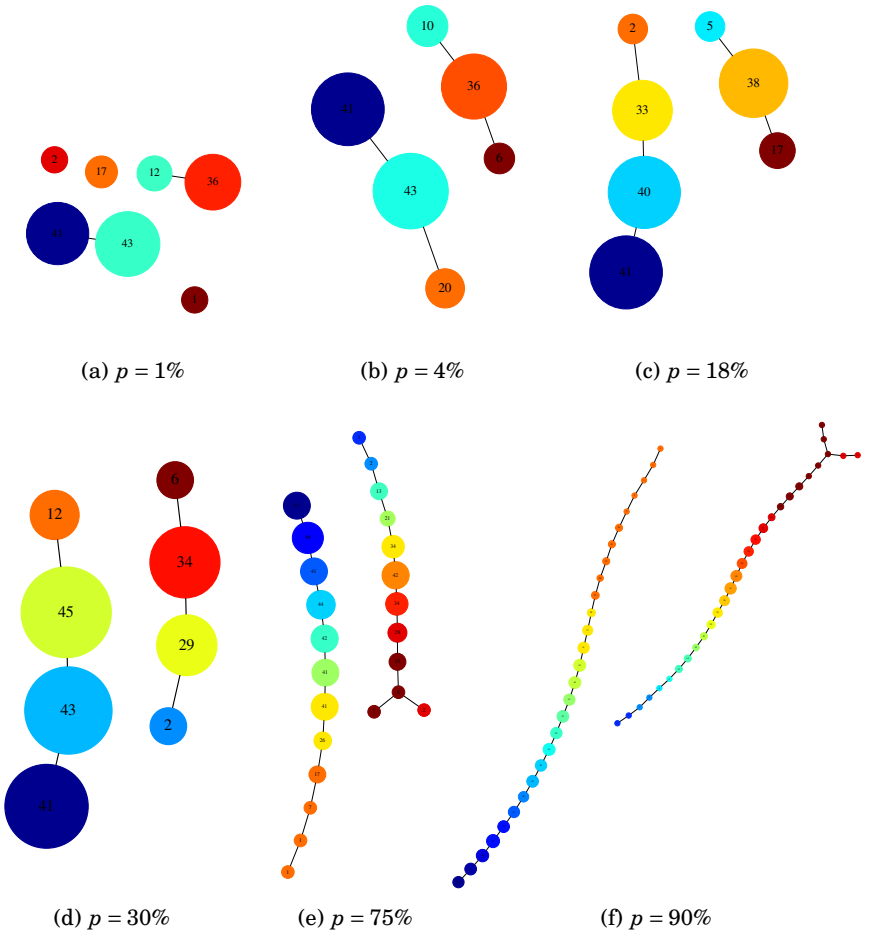


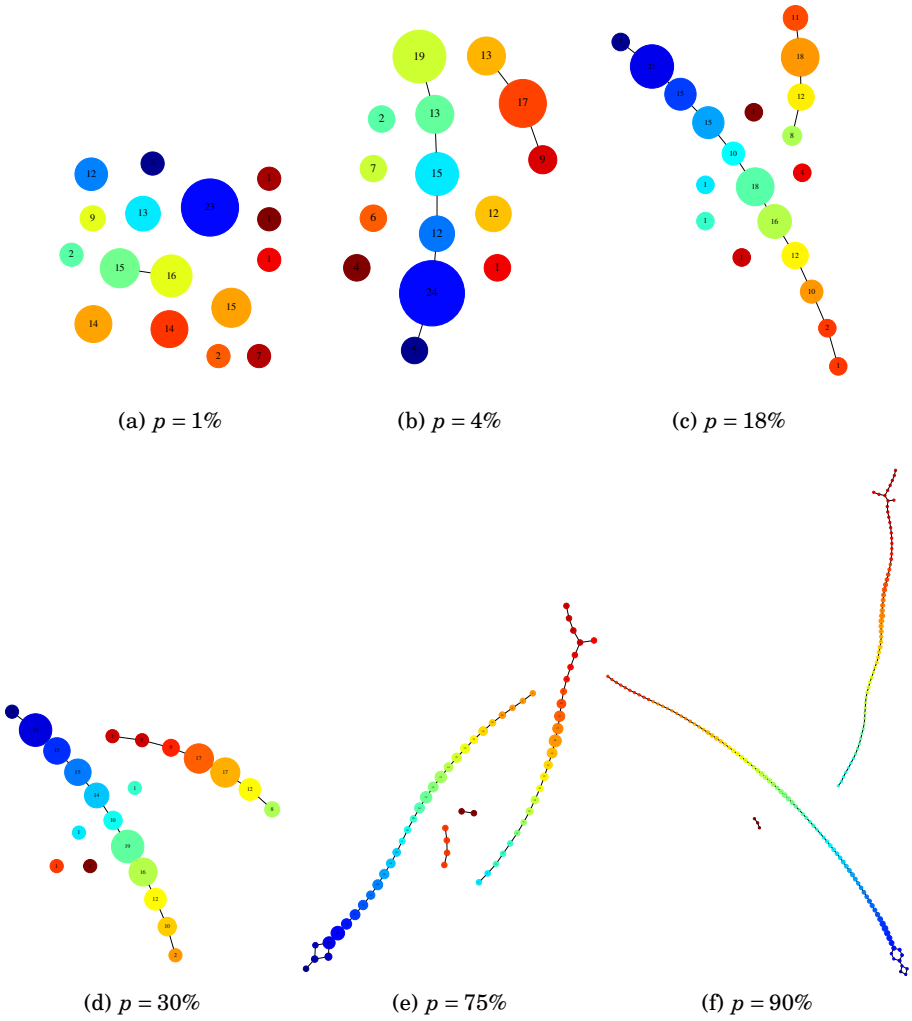
Figure 11.4.: Mapper of the Iris data with filter $f = ecc \cdot density$ and parameter space $\mathscr{U}(9,60)$. The text in each node says how many setosas, versicolors and virginicas there are in that node.



(a) $p = 1\%$ (b) $p = 4\%$ (c) $p = 18\%$

(d) $p = 30\%$ (e) $p = 75\%$ (f) $p = 90\%$

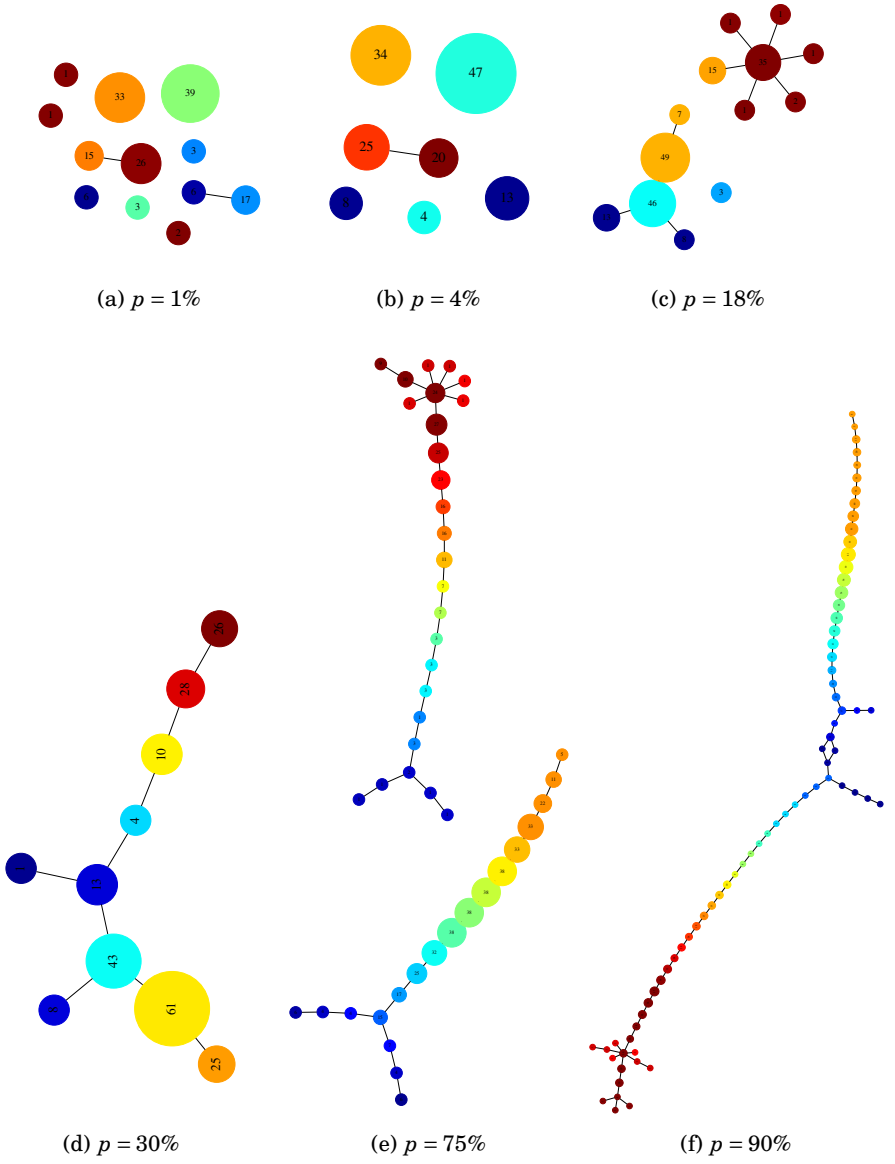
Figure 11.5.: Mapper output of the iris data with the fixed parameter $l = 0.3$ and different values of p. The output for $p = 60\%$ is shown in Figure 11.4.



(a) $p = 1\%$ (b) $p = 4\%$ (c) $p = 18\%$

(d) $p = 30\%$ (e) $p = 75\%$ (f) $p = 90\%$

Figure 11.6.: Mapper output of the iris data with the fixed parameter $l = 0.1$ and different values of p.



(a) $p = 1\%$ (b) $p = 4\%$ (c) $p = 18\%$

(d) $p = 30\%$ (e) $p = 75\%$ (f) $p = 90\%$

Figure 11.7.: Mapper output of the iris data using 3-means clustering with the fixed parameter $l = 0.1$ and different values of p. Note that there is one component in Figure 11.7d and two components in Figure 11.7e. This could not have happened with a functorial clustering algorithm.

11.3. Conclusions

We have now done an analysis of the Iris data set with different clustering methods, with a functorial Mapper and with a non-functorial Mapper. We now summarize our thoughts

The correct taxonomy of the Iris flowers suggest that there are three clusters in the data, but we saw in the introduction to this chapter that this is uncertain. Some of the clustering algorithms were able to cluster the data in three, but this is not surprising since for these methods we had to specify the amount of clusters beforehand and we set the number of clusters to three. The functorial Mapper quite consistently suggests that there are two clusters in the data and gave us little reason to believe that there might in actuality be three clusters there. From the non-functorial Mapper we can draw no conclusions at all.

There seems to be a benefit to Mapper that one can run the algorithm with different parameter values and with different filters. We argued that this makes Mapper more resistant to spurious behaviour based on a single result, because one will always rerun the algorithm initialized differently. Only features which persist through many of the different runs of Mapper should be considered valid. We argued further that a good scheme for choosing parameter values is to consider several covers $\mathscr{U}_i$ of the parameter space for $i \in \mathbb{N}$, with the property that $\mathscr{U}_i$ refines $\mathscr{U}_{i+1}$. With a functorial Mapper the maps induced from $\mathscr{U}_i \to \mathscr{U}_{i+1}$ suggest that the chains $\mathscr{U}_i$ is a good place to search for persistent topological features, and indeed, in the chains of covers which we studied we found clear and persistent features. These features disappeared with the non-functorial Mapper, further backing up our claim.

We must be careful, though, and ask ourself: The geometric features which we found to be persistent, were they persistent *because* we studied refining chains of covers? If, for instance, we have a randomly generated point cloud and run Mapper with respect to each of the covers of a chain $\mathscr{U}_i \to \mathscr{U}_{i+1}$, would we then observe persistent geometric features? If so, we must reconsider the way we use functorial clustering.

In doing these experiments we also felt the need for finding a better parametrized family of covers of an interval than the one presented in Section 7.3. Our belief was that for a fixed l and for p_0 and p_1 with $p_0 \leq p_1$ we would have that $\mathscr{U}(l, p_0)$ refines $\mathscr{U}(l, p_1)$. This is not the case, as for example the covers $\mathscr{U}(0.5, 50\%)$ and $\mathscr{U}(0.5, 33\%)$ of the unit interval shows.

We therefore had to throw away many of the old computations we had done, and calculate some new ones. We found that for $l = 0.1$ and $l = 0.3$ the series of overlap percentages $p = 1\%, 4\%, 18\%, 30\%, 75\%, 90\%$ we *do* have that $\mathscr{U}(l, p_i)$ refines $\mathscr{U}(l, p_{i+1})$, so these are the calculations we have presented. We therefore need to find a family of covers of an interval which is parametrized by some real parameter.

Afterthoughs

This is the rather abrupt end to the Master's thesis. We would have liked to add many more numerical experiments to study various properties of the Mapper, but we must content ourself with what we managed to include before the deadline.

We hope, however, that the reader has gotten enough of an impression of the algorithm to be motivated to study it further, because we find that it fills a hole in the current catalog of statistical methods. The biggest upside to Mapper is something which we have not yet said anything about — Mapper is a fun to use! It is a real delight to load a new dataset and to examine the resulting plots, and with better software it could have been even more fun. Our most immediate idea is to have Mapper integrated into a software suite (like the open-source *R*) such that one could do classical statistical methods like hypothesis testing directly in the Mapper software. One could for instance right-click on a node and get a drop-down menu asking you if you want to test the points in this node for statistical significance. The possibilities are endless!

Bibliography

[1] Graphviz, an open source graph visualization software. graphviz.org.

[2] Shai Ben-David, Ulrike von Luxburg, and Dávid Pál. A sober look at clustering stability. In *Proceedings of the 19th annual conference on Learning Theory*, COLT'06, pages 5–19, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-35294-5, 978-3-540-35294-5. doi: 10.1007/11776420_4. URL http://dx.doi.org/10.1007/11776420_4.

[3] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

[4] Gunnar Carlsson. "mapper for mapping", video lecture. http://www.ima.umn.edu/videos/?id=868, 2009.

[5] Gunnar Carlsson and Facundo Mémoli. Persistent clustering and a theorem of j. kleinberg, 2008. URL http://comptop.stanford.edu/preprints/clust-functorial.pdf.

[6] Patrik D'haeseleer. How does gene expression clustering work? *Nature Biotechnology*, 23(12):1499–1501, December 2005. ISSN 1087-0156. URL http://dx.doi.org/10.1038/nbt1205-1499.

[7] Daniel Dugger. A primer on homotopy colimits. http://www.uoregon.edu/~ddugger/hocolim.pdf, 2008.

[8] Martin Ester, Hans-peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *Computer*, 1996(6):226–231. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:A+Density-Based+Algorithm+for+Discovering+Clusters+in+Large+Spatial+Databases+with+Noise#0.

[9] R A Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936. URL `http://digital.library.adelaide.edu.au/dspace/handle/2440/15227`.

[10] A. Hatcher. *Algebraic topology*. Cambridge University Press, 2002. ISBN 9780521795401. URL `http://books.google.com/books?id=BjKs86kosqgC`.

[11] A. Jain, M. Murty, and P. Flynn. Data clustering: A review. *ACM Computing Survey*, 31(3):264–323, 1999.

[12] Ulrike Von Luxburg and Shai Ben-david. Towards a statistical theory of clustering. In *In PASCAL workshop on Statistics and Optimization of Clustering*, 2005.

[13] John Milnor. The geometric realization of a semi-simplicial complex. *The Annals of Mathematics*, 65(2):357–362, 1957. URL `http://www.jstor.org/stable/1969967`.

[14] Arnold J. Levine Monica Nicolau and Gunnar Carlsson. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. 2011. doi: 10.1073/pnas.1102826108.

[15] Graeme Segal. Classifying spaces and spectral sequences. *Publications Mathematiques De L Ihes*, 34:105–112, 1968. doi: 10.1007/BF02684591.

[16] Gurjeet Singh. Mapper: partial clustering in matlab. `http://comptop.stanford.edu/programs`, 2009.

[17] Gurjeet Singh, Facundo Memoli, and Gunnar Carlsson. Topological Methods for the Analysis of High Dimensional Data Sets and 3D Object Recognition. `comptop.stanford.edu/preprints/mapperPBG.pdf`.

[18] Jian; Huang Xuhui; Bowman Gregory R.; Singh Gurjeet; Lesnick Michael; Guibas Leonidas J.; Pande Vijay S.; Carlsson Gunnar Yao, Yuan; Sun. Topological methods for exploring low-density states in biomolecular folding pathways. *Journal of Chemical Physics*, 130. doi: 10.1063/1.3103496.