Topological Data Analysis with Julia

G. Vituri

2023-09-01

Table of contents

Preface

> "[O]f all the several ways of beginning a book which are now in practice throughout the known world, I am confident my own way of doing it is the best—I'm sure it is the most religious—for I begin with writing the first sentence—and trusting to Almighty God for the second."
> — Laurence Sterne, in "The Life and Opinions of Tristram Shandy, Gentleman"

Welcome!

This is the first draft of **"Topological data analysis with Julia"**.

The secret knowledge of Topological Data Analysis (TDA, for short) is spread in hundreds of papers and a few books. None, however, gives a consistent treatment of theory and examples with code. Code is essential to transform theory into real data analysis.

This book tries to fill this gap. In it, we will outline the main methods used to analyse data with topology, and try to give some non-trivial examples. Besides, it is a healthy way I found to practice Julia and study TDA again.

The readers who are afraid of Mathematics are urged to at least understand the intuitive notions of the definitions and results presented here. This is why I will give many informal descriptions of the ideas and objects before formalising them. Keep in mind, however, that Mathematics is the language that best describes abstractions and the use of logic, and **you only can learn a language by using it**.

This book will teach the basics of topology needed to understand TDA, and in the examples will give some directions on data analysis; but unfortunately we will not teach you Julia. For that, there are many excellent resources. See, for example:

- Think Julia
- Julia for Optimization and Learning
- Data Science in Julia for Hackers

1 Introduction

"The unexpected elation with which I had talked about mathematics had suddenly evaporated, and I sat beside him, feeling the weight of my own body, its unnecessary size. Outside of mathematics we had nothing to say to each other, and we both knew it. Then it occurred to me that the emotion with which I had spoken of the blessed role of mathematics on the voyage was a deception. I had been deceiving myself with the modesty, the serious heroism of the pilot who occupies himself, in the gaps of the nebulae, with theoretical studies of infinity. Hypocrisy. For what had it been, really? If a castaway, adrift for months at sea, has a thousand times counted the number of wood fibers that make up his raft, in order to keep sane, should he boast about it when he reaches land? That he had the tenacity to survive? And what of it? Who cared? Why should it matter to anyone how I had filled my poor brain those ten years, and why was that more important than how I had filled my stomach?"
— Stanisław Lem, in "Return from the stars"

Part I

Classical topology (or: topology without data nor analysis)

2 Topology

> "[…] mathematics stands above everything. The works of Abel and Kronecker are as good today as they were four hundred years ago, and it will always be so. New roads arise, but the old ones lead on. They do not become overgrown. There… there you have eternity. Only mathematics does not fear it. Up there, I understood how final it is. And strong. There was nothing like it. And the fact that I had to struggle was also good. I slaved away at it, and when I couldn't sleep I would go over, in my mind, the material I had studied that day."
> — Stanisław Lem, in "Return from the stars"

In this chapter we will define topological spaces and some of its properties, the notion of homeomorphism and so on.

2.1 What is topology?

Topology is the study of topological spaces and its properties. Unfortunately, this is not enough to close this chapter.

A topological space can be thought of as a set with little "scales" called *open sets*. Each of these open sets represent the notion of local *continuity* or *connectivity* that we sense when we look at an object: the feeling that it is just "one single piece of a thing", like the scales in a fish. We can stretch these scales, but can't rip them apart. In a sense, topological spaces are elastic. Because of that, there is an old joke that a topologist can't distinguish between a cup and a donut.

More formally,

Definition 2.1. A topological space is a pair (X, τ) where

- X is a set;
- τ is a set of subsets of X, that is: each $U \in \tau$ is a set $U \subseteq X$.

The set τ has the following properties:

[1] https://commons.wikimedia.org/wiki/File:Topology_joke.jpg

Figure 2.1: A continuous transformation of a cup to a donut. A true topologist will try to bite all these cups. Source: Wikipedia.[1]

- the intersection of a finite number of open sets is also an open set;
- the union of an arbitrary (even infinity) number of open sets is also an open set;
- the $\emptyset$ and X are open sets.

The set τ is called a *topology* on X.

We often omit τ and simply write "a topological space X". Sometimes we omit the "topological" too. We love omiting!

Definition 2.2. The set of all subsets (or "parts") of X is denoted by $P(X)$, that is,

$$P(X) = \{U \mid U \subseteq X\}.$$

Given a set X, there is a easy way to generate a topology on it:

Definition 2.3. Let X be a set and $S \subseteq P(X)$. Define τ $\{\emptyset, X\}$ united with finite intersections and arbitrary unions of elements of S. The pair (X, τ) is a topological space, and τ is said to be "generated by S". We also say that S is a generator set for τ.

Don't let the abstractions hurt you! Whenever you see a new definition or theorem, think of an example of objects that fit in it. One very useful example is the following:

Example 2.1. The standard topology on the set $\mathbb{R}$ of real numbers is generated by the open intervals $(a, b) \subseteq \mathbb{R}$.

In the above case, an open set is a set in which one can always "walk a little more without reaching the end of the set". More precisely, given $x \in (a, b)$ there is always a small enough $\epsilon > 0$ such the interval $(x - \epsilon, x + \epsilon)$ is contained in (a, b).

!![fora de lugar] When an open set U cannot be written as the disjoint union of two open sets, we say that U is *connected*. These connected open sets give a notion of "being a single piece".

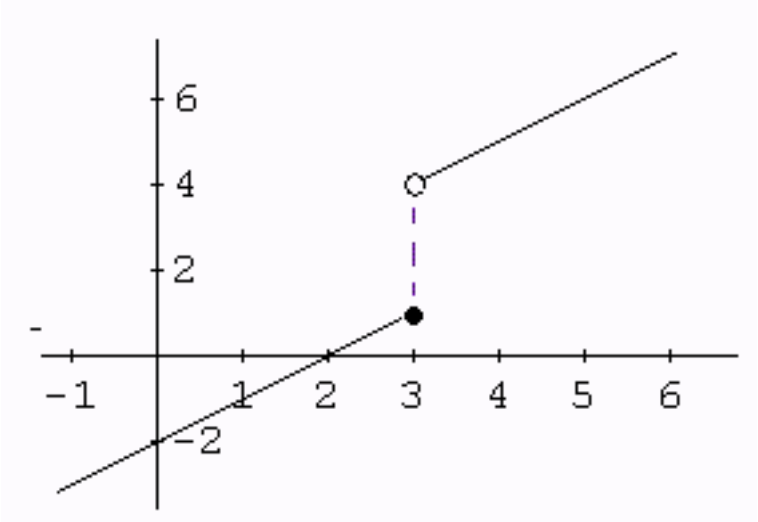


Figure 2.2: An example of a discontinuous function. Source: byjus.com[3]

2.2 Continuity

We learn in Calculus that a function $f : \mathbb{R} \to \mathbb{R}$ is *continuous* when we can draw its graph "without lifting the pencil from the paper". Formalizing this notion of continuity took centuries and made many mathematicians go crazy.[2] Let's try to formalize what is "to lift the pencil from the paper" with the following example:

We notice that no matter how close to 3 we choose some $x \in \mathbb{R}$, its image $f(x)$ will end up being far away from $f(3)$. A bit more precisely, there is a small interval around $f(3)$, say $(f(3) - \epsilon, f(3) + \epsilon)$, such that no matter how small we chose an interval around 3, say $(3 - \delta, 3 + \delta)$ we will have that

$$f(x) \notin (f(3) - \epsilon, f(3) + \epsilon), \text{ for some } x \in (3 - \delta, 3 + \delta).$$

!![checar se faz sentido] This is equivalent to the fact that the inverse image of $(f(3)-\epsilon, f(3)+\epsilon)$ is *not* an open set: if it were an open set, we could always "walk a little" on this interval, and still be sent by f to $(f(3) - \epsilon, f(3) + \epsilon)$.

Inspired by that, we define the following in the much more general context of topological spaces:

Definition 2.4. Given two topological spaces (X, τ) and (Y, σ), a function $f : X \to Y$ is said to be *continuous* if the inverse image of any open set of Y is also an open set of X. In symbols:

[2] I just invented this. Maybe some of them really got crazy, who knows?
[3] https://byjus.com/maths/discontinuity/

$$\forall\, V \in \sigma,\ f^{-1}(V) \in \tau.$$

Remark. Negating what we obtained in our example above leads us to the definition of continuity of real functions, and to the most terrifying single line of math of first-year students: a function $f : \mathbb{R} \to \mathbb{R}$ is *continuous* if for every point $a \in \mathbb{R}$ we have

$$\forall \epsilon > 0, \exists \delta > 0 \text{ such that } x \in (a-\delta, a+\delta) \text{ implies } f(x) \in (f(a)-\epsilon, f(a)+\epsilon),$$

which is equivalent to $\lim_{x\to a} f(x) = a$.

2.3 The notion of "sameness" in topology

Let's digress a little.

A topological space is completely define by a set X and its open sets τ. If we just "rename" the elements of X, then it is intuitive that this new topological space is as similar to X as possible; in fact, they are indistinguishable from each other: every topological property (ie: properties related to open sets) that the first has, the second also has.

Renaming points is just a informally way to say "bijection of sets". So, two topological spaces X and Y are "the same thing, topologically speaking" if there is a bijection $f : X \to Y$ such that $U \subseteq X$ is open if and only if $f(X) \subseteq Y$ is open. More precisely,

Definition 2.5. We say that two topological spaces X and Y are *homeomorphic* if there exists a continuous bijection $f : X \to Y$ such that f^{-1} is also continuous. In this case, f is called an *homeomorphism*.

3 Simplicial complexes

"Perfection does not exist; to comprehend it is the triumph of human intelligence; to desire to possess it, the most dangerous of follies."
— Alfred de Musset, in "The confession of a child of the century"

4 Simplicial homology

Introduzir homologia como modo de contar buracos. Ideia intuitiva, depois formalização

Part II

Topological data analysis (or: topology meets reality)

5 Persistent homology

"The essence of this architecture is movement synchronized towards a precise objective. We observe a fraction of the process, like hearing the vibration of a single string in an orchestra of supergiants. We know, but cannot grasp, that above and below, beyond the limits of perception or imagination, thousands and millions of simultaneous transformations are at work, interlinked like a musical score by mathematical counterpoint. It has been described as a symphony in geometry, but we lack the ears to hear it."
— Stanisław Lem, in "Solaris"

Part III

Case studies

6 Classifying hand-written digits

> "Thank God for giving you a glimpse of heaven, but do not imagine yourself a bird because you can flap your wings."
> — Alfred de Musset, in "The confession of a child of the century"

In this tutorial, we will try to classify hand-written digits using the tools seen in previous chapters.

6.1 Loading packages

```
import MLDatasets
using Images, Makie, CairoMakie
using Distances
using Ripserer, PersistenceDiagrams
using StatsBase: mean
import Plots;
using DataFrames, FreqTables, PrettyTables
using Flux, ProgressMeter
```

6.2 The dataset

MNIST is a dataset consisting of 70.000 hand-written digits. Each digit is a 28x28 grayscale image, that is: a 28x28 matrix of values from 0 to 1. To get this dataset, run

```
n_train = 10_000
mnist_digits, mnist_labels = MLDatasets.MNIST(split=:train)[:];
mnist_digits = mnist_digits[:, :, 1:n_train]
mnist_labels = mnist_labels[1:n_train];
```

If the console asks you to download some data, just press `y`.

Notice that we only get the first `n_train` images so this notebook doesn't take too much time to run. You can increase `n_train` to 60000 if you like to live dangerously and have enough RAM memory.

Next, we transpose the digits and save them in a vector

```
figs = [mnist_digits[:, :, i]' |> Matrix for i  1:size(mnist_digits)[3]];
```

The first digit, for example, is the following matrix:

```
figs[1]
```

```
28×28 Matrix{Float32}:
 0.0  0.0  0.0  0.0  0.0       0.0       …  0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0       …  0.498039  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.25098   0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0       …  0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0

 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0       …  0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.215686  0.67451      0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.533333  0.992157     0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0       …  0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0       0.0          0.0       0.0  0.0  0.0  0.0
```

We can see a mosaic with the first 10^2 digits

```
n = 10
figs_plot = [fig .|> Gray for fig in figs[1:n^2]]
mosaicview(figs_plot, nrow = n, rowmajor = true)
```

6.3 Preparing for war

What topological tools can be useful to distinguish between different digits?

Persistence homology with Vietoris-Rips filtration won't be of much help: all digits are connected, so the 0-persistence is useless; for the 1-dimensional persistence,

- 1, 3, 5, 7 do not contain holes;
- 2 and 4 sometimes contain one hole (depending on the way you write it);
- 0, 6, 9 contain one hole each;
- 8 contains two holes.

What if we starting chopping the digits with sublevels of some functions? The *excentricity* function is able to highlight edges. Doing a sublevel filtration with the excentricity function will permit us to separate digits by the amount of edges they have. So 1 and 3 and 7, for example, will have different persistence diagrams.

6.3.1 From square matrices to points in the plane

In order to calculate the excentricity, we need to convert the "image digits" (28x28 matrices) to points in $\mathbb{R}^2$ (matrices with 2 columns, one for each dimension, which we will call *pointclouds*). A simple function can do that:

```
function img_to_points(img, threshold = 0.3)
    ids = findall(x -> x >= threshold, img)
    pts = getindex.(ids, [1 2])
end;
```

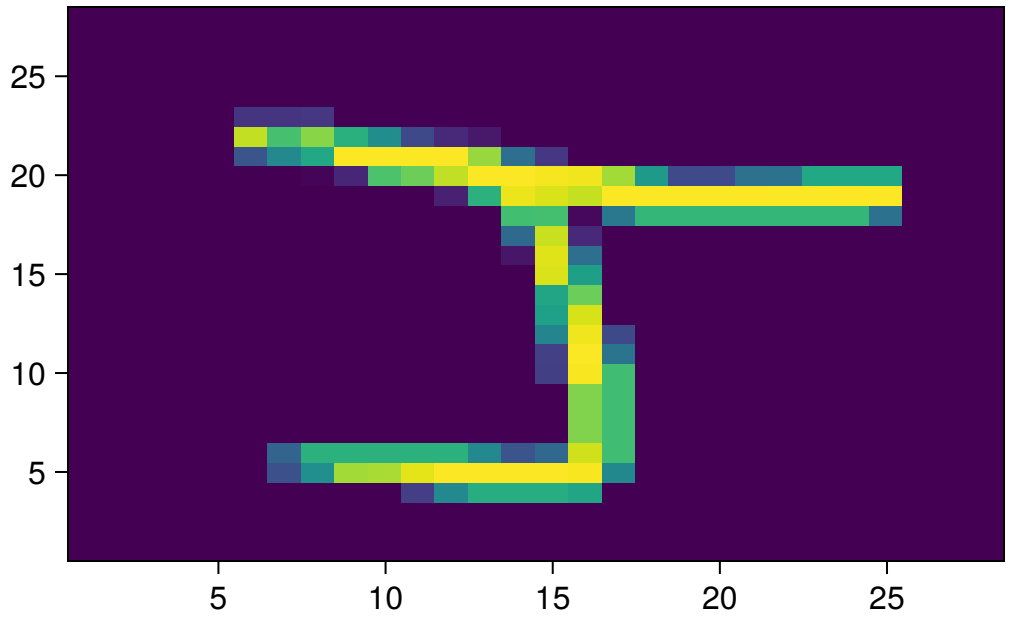
Notice that we had to define a threshold: coordinates with values less than the threshold are not considered.

Let's also define a function to plot a digit:

```
function plot_digit(fig, values = :black)
    pt = img_to_points(fig)
    f = Figure();
    ax = Makie.Axis(f[1, 1], autolimitaspect = 1, yreversed = true)
    scatter!(ax, pt[:, 2], pt[:, 1]; markersize = 40, marker = :rect, color = values)
    if values isa Vector{<:Real}
        Colorbar(f[1, 2])
    end
    f
end;
```

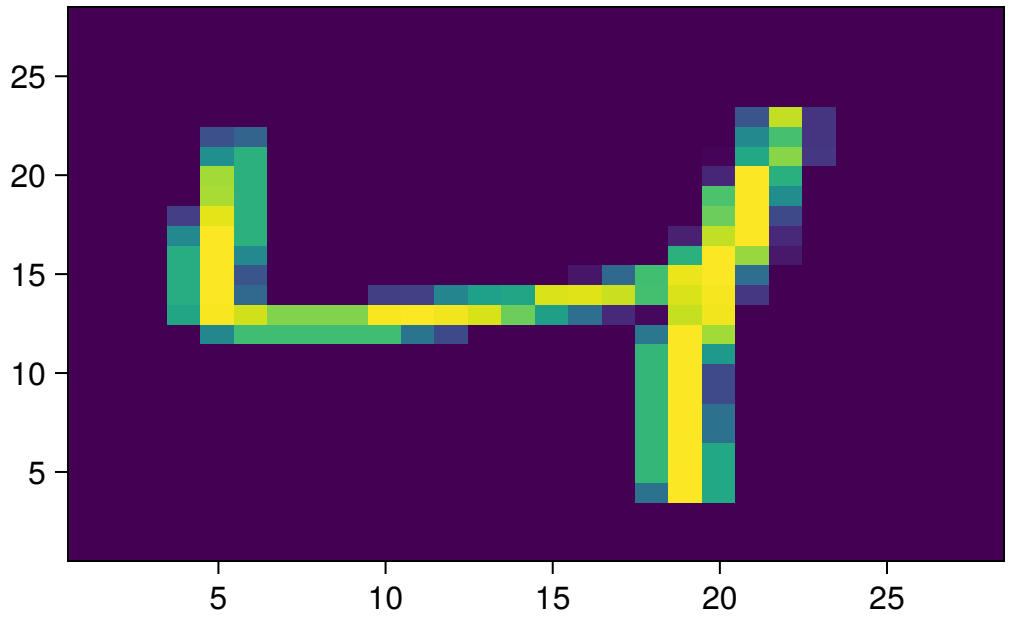
We can see that it works as expected

```
fig = figs[3]
heatmap(fig)
```



but the image is flipped. This is easily fixed:

```
heatmap(fig |> rotr90)
```



6.3.2 Excentricity

Getting into details: the *excentricity* of a metric space (X, d) is a measure of how far a point is from the "center". It is defined as follows for each $x \in X$:

$$e(x) = \sum_{y \in X} \frac{d(x, y)}{N}$$

where N is the amount of points of X.

Define a function that takes a digit in $\mathbb{R}^2$ and return the excentricity as an 28x28 image

```
function excentricity(fig)
    pt = img_to_points(fig)
    dists = pairwise(Euclidean(), pt')
    excentricity = [mean(c) for c  eachcol(dists)]
    exc_matrix = zeros(28, 28)

    for (row, (i, j))  enumerate(eachrow(pt))
        exc_matrix[i, j] = excentricity[row]
    end
```

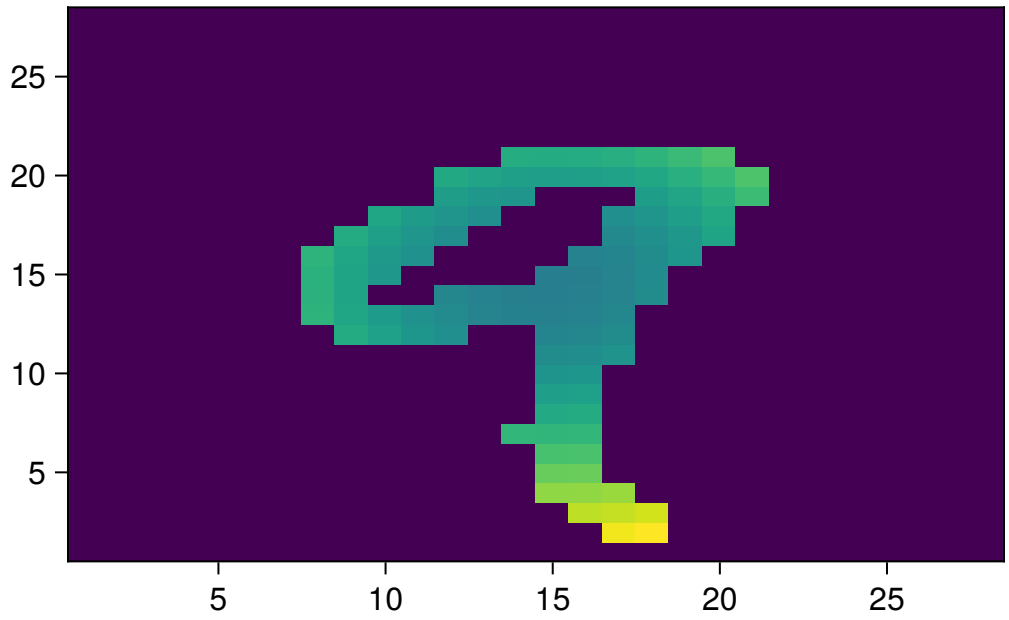
```
    return exc_matrix
end;
```

We store all the excentricities in the `excs` vector

```
excs = excentricity.(figs);
```

and plot a digit with it's corresponding excentricity

```
i = 5
fig = figs[i]
exc = excs[i]
heatmap(exc |> rotr90)
```



Looks good! Time to chop it.

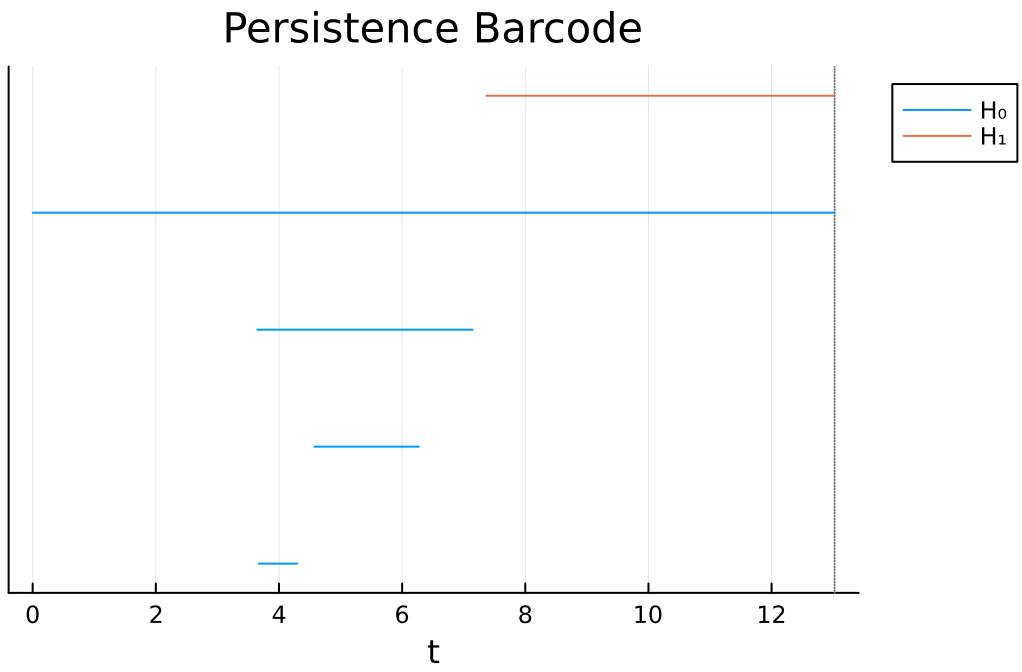
6.3.3 Persistence images

Now we calculate all the persistence diagrams using sublevel filtration. This can take some seconds. Julia is incredibly fast, but does not perform miracles (yet!).

```
pds = map(excs) do ex
    m = maximum(ex)
    ex = m .- ex
    ripserer(Cubical(ex), cutoff = 0.5)
end;
```

We check the first one

```
pd = pds[i]
pd |> barcode
```

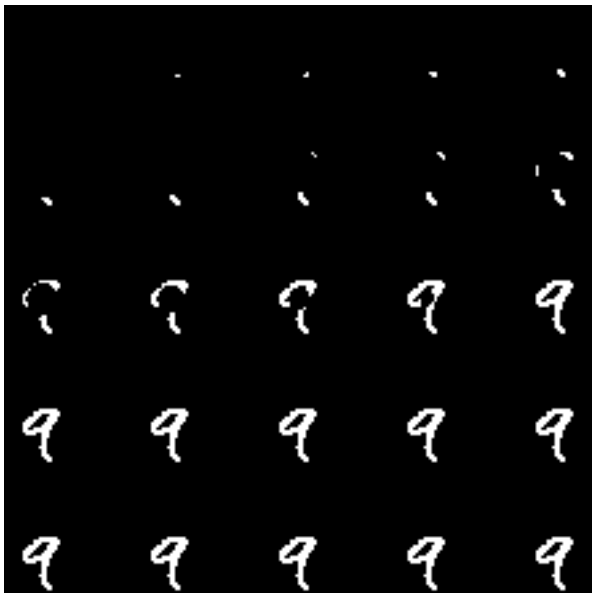


Compare it with the corresponding heatmap above. There are 3 main edges (and one really small one). It seems ok!

We can see the "step-by-step" creation of these connected components in the following mosaic.

```
r = range(minimum(exc), maximum(exc), length = 25) |> reverse
figs_filtration = map(r) do v
    replace(x -> x   v ? 0 : 1, exc) .|> Gray
end
```

```
mosaicview(figs_filtration..., rowmajor = true, nrow = 5, npad = 20)
```



Now we create the persistence images of all these barcodes in dimension 0 and 1. We pass the entire collection of barcodes to the `PersistenceImage` function, and it will ensure that all of them are comparable (ie. are on the same grid).

```
pds_0 = pds .|> first
pds_1 = pds .|> last
imgs_0 = PersistenceImage(pds_0; sigma = 1, size = 8)
imgs_1 = PersistenceImage(pds_1; sigma = 1, size = 8);
```

The persistence images look ok too:

```
Plots.plot(
    barcode(pds[i])
    , Plots.plot(pds[i]; persistence = true)
    , Plots.heatmap(imgs_0(pds[i][1]); aspect_ratio=1)
    ,  Plots.heatmap(imgs_1(pds[i][2]); aspect_ratio=1)
    , layout = (2, 2)
    )
```

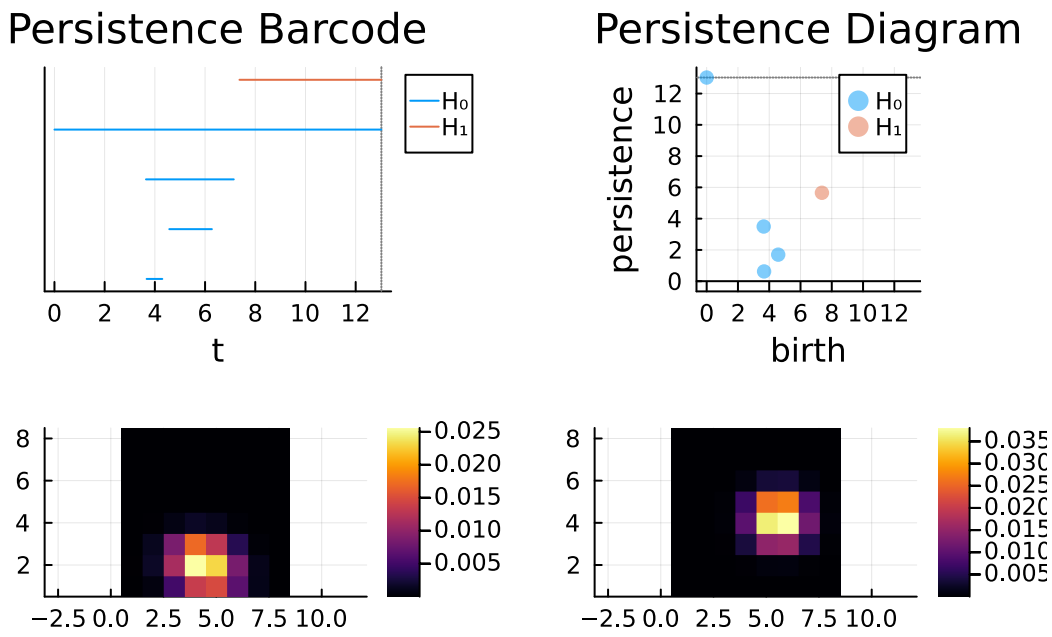


Figure 6.1: Top left: the barcode of a digit with respect to sublevels using the excentricity function. Top right: the corresponding persistence diagram. Bottom: 0 and 1 dimensional persistence images. They create a pixelated view of the persistence diagram, using a gaussian blur.

6.4 Fitting a model

In order to use these persistence images in a machine learning model, we first need to vectorize them, ie, transform them into a vector. Machine learning models love vectors! The easist way is to just concatenate the persistence images as follows:

```
function concatenate_pds(imgs_0, pds_0, imgs_1, pds_1)
    persims = [
        [vec(imgs_0(pds_0[i])); vec(imgs_1(pds_1[i])) ] for i in 1:length(pds)
        ]

    X = reduce(hcat, persims)'
    X
end

X = concatenate_pds(imgs_0, pds_0, imgs_1, pds_1)
```

```
y = mnist_labels .|> string;
```

We can see that `X` is a matrix with 10000 rows (the amount of digits) and 128 columns (the persistence images concatenated).

It was also important to convert the `mnist_labels` to strings, because we want to classify the digits (and not do a regression on them).

We now have a vector for each image. What can we do? We need a model that takes a large vector of numbers and try to predict the digit. Neural networks are excellent in finding non-linear relations on vectors. Let's try one!

Create the layers

```
function nn_model(X)
  model = Chain(
      Dense(size(X)[2] => 64)
      ,Dense(64 => 10)
  )
end

model = nn_model(X)
```

```
Chain(
  Dense(128 => 64),                     # 8_256 parameters
  Dense(64 => 10),                      # 650 parameters
)                   # Total: 4 arrays, 8_906 parameters, 35.039 KiB.
```

the loader

```
target = Flux.onehotbatch(y, 0:9 .|> string)
loader = Flux.DataLoader((X' .|> Float32, target), batchsize=32, shuffle=true);
```

the optimiser

```
optim = Flux.setup(Flux.Adam(0.01), model);
```

and train it

```
@showprogress for epoch in 1:100
    Flux.train!(model, loader, optim) do m, x, y
        y_hat = m(x)
        Flux.logitcrossentropy(y_hat, y)
```

```
    end
end;
```

The predictions can be made with

```
pred_y = model(X' .|> Float32)
pred_y = Flux.onecold(pred_y, 0:9 .|> string);
```

And the accuracy

```
accuracy = sum(pred_y .== y) / length(y)
accuracy = round(accuracy * 100, digits = 2)
println("The accuracy on the train set was $accuracy %!")
```

```
The accuracy on the train set was 68.0 %!
```

Not bad, taking into account that we only used the excentricity sublevel filtration.

The confusion matrix is the following:

```
tbl = freqtable(y, pred_y)
```

```
10×10 Named Matrix{Int64}
Dim1   Dim2      0     1     2     3     4     5     6     7     8     9

0              926     0    20     1     1    19    20     2     5     7
1                1  1081     5     0     4    16     0    20     0     0
2               22    18   547     1    93   120    26    74    54    36
3               11    43    57   238    24   504    12   128     9     6
4                1     2    75     1   749    17     3   120     4     8
5               12    47    34    15     7   612    19   103     3    11
6               17     2     9     2     7     9   794    66     1   107
7                2    47    31     3   122    59    50   754     1     1
8               21     4   129     2    24     5    12     9   700    38
9               13     1    20     0    15    20   402   104     4   399
```

Calculating the proportion of prediction for each digit, we get

```
round2(x) = round(100*x, digits = 1)
```

```
function prop_table(y1, y2)
    tbl = freqtable(y1, y2)
    tbl_prop = prop(tbl, margins = 1) .|> round2
    tbl_prop
end

tbl_p = prop_table(y, pred_y)
```

```
10×10 Named Matrix{Float64}
Dim1   Dim2      0     1     2     3     4     5     6     7     8     9

0             92.5   0.0   2.0   0.1   0.1   1.9   2.0   0.2   0.5   0.7
1              0.1  95.9   0.4   0.0   0.4   1.4   0.0   1.8   0.0   0.0
2              2.2   1.8  55.2   0.1   9.4  12.1   2.6   7.5   5.4   3.6
3              1.1   4.2   5.5  23.1   2.3  48.8   1.2  12.4   0.9   0.6
4              0.1   0.2   7.7   0.1  76.4   1.7   0.3  12.2   0.4   0.8
5              1.4   5.4   3.9   1.7   0.8  70.9   2.2  11.9   0.3   1.3
6              1.7   0.2   0.9   0.2   0.7   0.9  78.3   6.5   0.1  10.6
7              0.2   4.4   2.9   0.3  11.4   5.5   4.7  70.5   0.1   0.1
8              2.2   0.4  13.7   0.2   2.5   0.5   1.3   1.0  74.2   4.0
9              1.3   0.1   2.0   0.0   1.5   2.0  41.1  10.6   0.4  40.8
```

We see that the biggest errors are the following:

```
function top_errors(tbl_p)
    df = DataFrame(
        Digit = Integer[]
        , Prediction = Integer[]
        , Percentage = Float64[]
        )

    for i = eachindex(IndexCartesian(), tbl_p)
        push!(df, (i[1]-1, i[2]-1, tbl_p[i]))
    end

    filter!(row -> row.Digit != row.Prediction, df)
    sort!(df, :Percentage, rev = true)
    df[1:10, :]
end

df_errors = top_errors(tbl_p)
```

```
df_errors |> pretty_table
```

```
  Digit   Prediction   Percentage
Integer      Integer      Float64

      3            5         48.8
      9            6         41.1
      8            2         13.7
      3            7         12.4
      4            7         12.2
      2            5         12.1
      5            7         11.9
      7            4         11.4
      9            7         10.6
      6            9         10.6
```

6.4.1 The perils of isometric spaces

How to separate "6" and "9"? They are isometric! For some people, "2" and "5" are also isometric (just mirror on the x-axis). Functions that only "see" the metric (like the excentricity) will never be able to separate these digits. In digits, the position of the features is important, so let's add more slicing filtrations to our arsenal.

To avoid writing all the above code-blocks again, we encapsulate the whole process into a function

```
function whole_process(
    mnist_digits, mnist_labels, f
    ; imgs_0 = nothing, imgs_1 = nothing
    , dim_max = 1, sigma = 1, size_persistence_image = 8
    )
    figs = [mnist_digits[:, :, i]' |> Matrix for i  1:size(mnist_digits)[3]]

    excs = f.(figs);

    pds = map(excs) do ex
        m = maximum(ex)
        ex = m .- ex
        ripserer(Cubical(ex), cutoff = 0.5, dim_max = dim_max)
```

```
    end;

    pds_0 = pds .|> first
    pds_1 = pds .|> last

    if isnothing(imgs_0)
        imgs_0 = PersistenceImage(pds_0; sigma = sigma, size = size_persistence_image)
    end
    if isnothing(imgs_1)
        imgs_1 = PersistenceImage(pds_1; sigma = sigma, size = size_persistence_image)
    end

    persims = [
    [vec(imgs_0(pds_0[i])); vec(imgs_1(pds_1[i])) ] for i in eachindex(pds)
    ]

    X = reduce(hcat, persims)'
    y = mnist_labels .|> string

    return X, y, pds_0, pds_1, imgs_0, imgs_1
end;
```

We now create the sideways filtrations: from the side and from above.

```
set_value(x, threshold = 0.5, value = 0) = x   threshold ? value : 0

function filtration_sideways(fig; axis = 1, invert = false)

  fig2 = copy(fig)
  if axis == 2 fig2 = fig2' |> Matrix end

  for i   1:28
    if invert k = 29 - i else k = i end
    fig2[i, :] .= set_value.(fig2[i, :], 0.5, k)
  end

  fig2

end;
```

and calculate all 4 persistence diagrams. Warning: this can take a few seconds if you are using 60000 digits!

```
fs = [
    x -> filtration_sideways(x, axis = 1, invert = false)
    ,x -> filtration_sideways(x, axis = 2, invert = false)
    ,x -> filtration_sideways(x, axis = 1, invert = true)
    ,x -> filtration_sideways(x, axis = 2, invert = true)
]

ret = @showprogress map(fs) do f
    whole_process(
        mnist_digits, mnist_labels, f
        ,size_persistence_image = 8
    )
end;
```

We concatenate all the vectors

```
X_list = ret .|> first
X_all = hcat(X, X_list...);
```

and try again with a new model:

```
model = nn_model(X_all)

target = Flux.onehotbatch(y, 0:9 .|> string)
loader = Flux.DataLoader((X_all' .|> Float32, target), batchsize=64, shuffle=true);

optim = Flux.setup(Flux.Adam(0.01), model)

@showprogress for epoch in 1:50
    Flux.train!(model, loader, optim) do m, x, y
        y_hat = m(x)
        Flux.logitcrossentropy(y_hat, y)
    end
end;
```

Now we have

```
pred_y = model(X_all' .|> Float32)
pred_y = Flux.onecold(pred_y, 0:9 .|> string)

accuracy = sum(pred_y .== y) / length(y)
accuracy = round(accuracy * 100, digits = 2)
```

```
println("The accuracy on the train set was $accuracy %!")
```

```
The accuracy on the train set was 95.49 %!
```

which is certainly an improvement!

The proportional confusion matrix is

```
prop_table(y, pred_y)
```

```
10×10 Named Matrix{Float64}
Dim1   Dim2      0     1     2     3     4     5     6     7     8     9

0             98.6   0.0   0.2   0.5   0.1   0.2   0.1   0.0   0.3   0.0
1              0.0  97.5   0.5   0.5   0.1   0.3   0.0   1.1   0.0   0.0
2              0.0   0.7  85.2   2.4   0.0   8.2   0.3   3.1   0.0   0.1
3              0.0   0.3   0.5  94.3   0.0   1.4   0.0   3.5   0.0   0.1
4              0.0   0.2   0.4   0.0  97.3   0.1   0.2   0.6   0.5   0.6
5              0.0   0.7   1.9   2.4   0.0  94.0   0.7   0.1   0.1   0.1
6              0.0   0.2   0.3   0.1   0.2   0.7  98.4   0.1   0.0   0.0
7              0.0   1.6   0.7   1.4   0.2   0.6   0.1  95.2   0.0   0.3
8              0.0   0.1   0.6   0.1   1.0   0.2   0.2   0.1  96.1   1.6
9              0.0   0.1   0.0   0.1   0.4   0.3   0.2   1.0   0.0  97.9
```

6.5 Learning from your mistakes

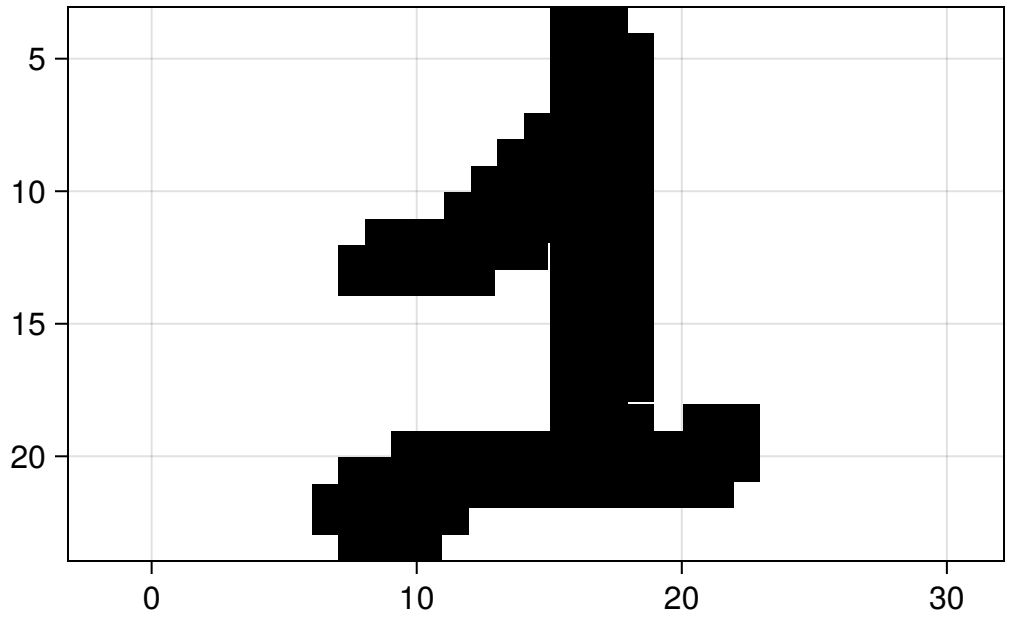
Let's explore a bit where the model is making mistakes. Collect all the errors

```
errors = findall(pred_y .!= y);
```

and plot the first 3

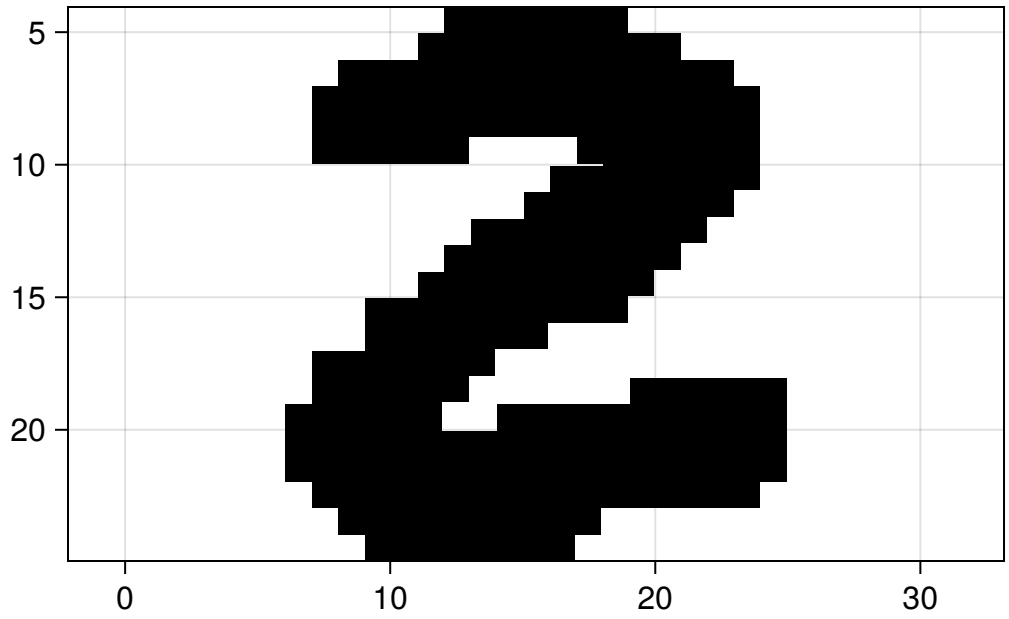
```
i = errors[1]
println("The model predicted a $(pred_y[i]) but it was a $(y[i])")
plot_digit(figs[i])
```

```
The model predicted a 2 but it was a 1
```



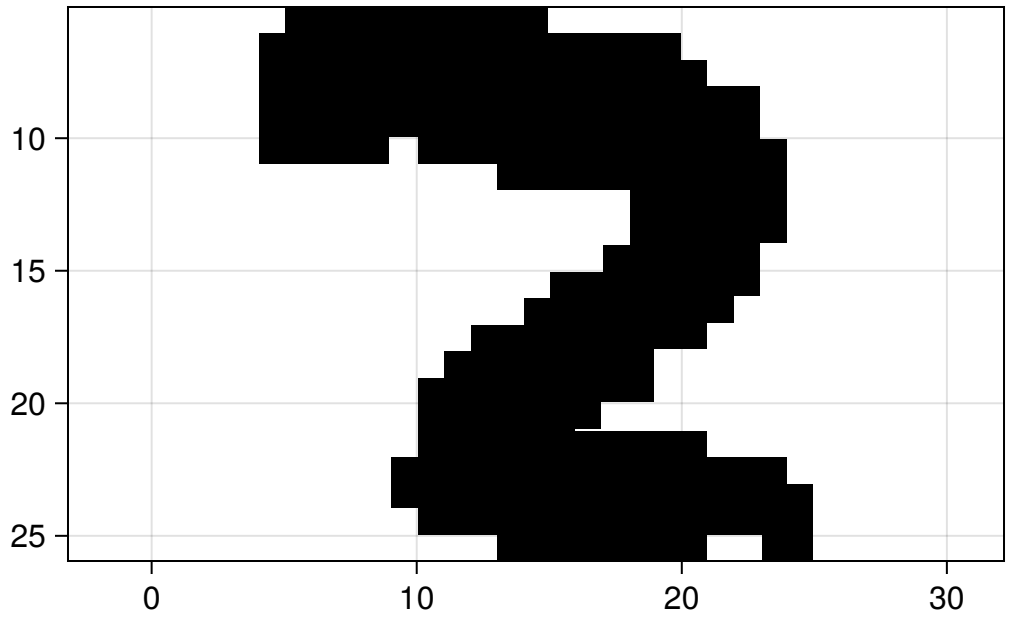

```
i = errors[2]
println("The model predicted a $(pred_y[i]) but it was a $(y[i])")
plot_digit(figs[i])
```

```
The model predicted a 5 but it was a 2
```




```
i = errors[3]
println("The model predicted a $(pred_y[i]) but it was a $(y[i])")
plot_digit(figs[i])
```

```
The model predicted a 5 but it was a 2
```



We can make a mosaic with the first 100 errors

```
n = 10
figs_plot = [figs[i] .|> Gray for i in errors[1:n^2]]
mosaicview(figs_plot, nrow = n, rowmajor = true)
```

Many of these digits are really ugly! This makes them hard to classify with our sublevel filtrations. Some other functions could be explored.

6.6 Getting new data

Now we want to see if our model really learned something, or if it just repeated what he saw in the training data. To check data, we need to get new data and calculate the accuracy of the same model on this new data.

```
n_test = 5_000
new_mnist_digits, new_mnist_labels = MLDatasets.MNIST(split=:test)[:];
new_mnist_digits = new_mnist_digits[:, :, 1:n_test]
new_mnist_labels = new_mnist_labels[1:n_test];
```

and obtaning `X` and `y` to feed the model

```
fs = [
    x -> excentricity(x)
```

```
    ,x -> filtration_sideways(x, axis = 1, invert = false)
    ,x -> filtration_sideways(x, axis = 2, invert = false)
    ,x -> filtration_sideways(x, axis = 1, invert = true)
    ,x -> filtration_sideways(x, axis = 2, invert = true)
]

ret = @showprogress map(fs) do f
    whole_process(
        new_mnist_digits, new_mnist_labels, f
        ,size_persistence_image = 8
    )
end;
```

Define our new `X` and `y`

```
new_X = ret .|> first
new_X = hcat(new_X...)
new_y = ret[1][2]

new_pred_y = model(new_X' .|> Float32)
new_pred_y = Flux.onecold(new_pred_y, 0:9 .|> string);
```

and calculate the accuracy:

```
accuracy = sum(new_pred_y .== new_y) / length(new_y)
accuracy = round(accuracy * 100, digits = 2)
println("The accuracy on the test data was $accuracy %!")
```

```
The accuracy on the test data was 87.4 %!
```

A bit less than the training set, but not so bad.

Let's check the confusion matrix

```
tbl = prop_table(new_y, new_pred_y)
```

```
10×10 Named Matrix{Float64}
Dim1   Dim2      0     1     2     3     4     5     6     7     8     9

0             94.1   0.0   0.9   0.2   0.7   0.2   1.3   0.4   1.5   0.7
1              0.0  96.0   1.1   0.4   0.2   0.9   0.2   1.2   0.2   0.0
```

2	0.9	0.6	77.4	4.2	0.6	11.7	1.3	2.1	1.1	0.2
3	0.0	1.2	5.0	84.6	0.0	4.4	0.2	4.0	0.4	0.2
4	0.0	0.6	1.4	0.4	93.4	1.0	0.2	0.8	0.4	1.8
5	0.2	0.7	10.7	4.8	1.1	78.3	0.4	1.3	1.8	0.7
6	1.3	0.2	3.0	0.6	1.1	2.6	87.7	0.0	3.0	0.4
7	0.2	2.1	2.9	3.9	5.1	1.4	0.0	83.6	0.0	0.8
8	1.0	0.0	5.3	0.2	0.2	0.6	0.0	0.0	89.6	3.1
9	0.0	0.2	0.6	0.2	4.0	0.6	0.4	2.1	3.3	88.7

6.7 Closing remarks

Even though we used heavy machinery from topology, at the end our persistence images were vectors that indicated the birth and death of edges. Apart from that, the only machine learning algorithm we used was a simple dense neural network to fit these vectors to the correct labels in a non-linear way. State-of-art machine learning models on the MNIST dataset usually can get more than 99% of accuracy, but they use some complicated neural networks with many layers, and the output prediction are hard to explain. These methods, however, are not excludent of each other: we can use the persistence images (and any other vectorized output from TDA) together with other algorithms.

A curious exercise to the reader is to check if a neural network with two parallel inputs - one for the digits images, followed by convolutional layers - other for the vector of persistence images, followed by dense layers can achieve a better result than the convolutional alone.

Part IV

Closing

References