SHAPE DETECTION
Topological data analysis

Ana julija Prešeren and Manca Strah

The aim of our project was to develop a model capable of differentiating between various geometric shapes using a topological data analysis approach.

1 Introduction

We generated point clouds representing different shapes in 3-dimensional space and used data from persistence diagrams to differentiate among them. However, some shapes are homotopy equivalent and we cannot distinguish them just based on the persistence diagrams, since they are similar. One example are a line segment and a rectangle. They both only have one connected component and zero holes. Another example are a sphere and an ellipsoid, both with one connected component and one cave. In order to distinguish among them, we modified the shapes by removing open balls from existing point clouds.

Removing an open ball from a point cloud that represents a line segment yields point cloud with two connected components and zero holes while removing an open ball from a rectangle (or flat disc) results in a point cloud with one connected component and one hole. Similarly, removing an open ball from an ellipsoid gives us two connected components while removing it from a sphere, the remaining point cloud still represents a sphere (with one connected component and one cave).

Modifying data as described, we were able to differentiate among homotopy equivalent shapes.

Lastly, we modified our model so that it was able to distinguish shapes in 4-dimensional space.

2 Model

In the `m_create_point clouds.py` script, functions for generation of point cloud representations of 8 different shapes are included. The shapes we chose to include are spheres, circles, line segments, tori, flat discs, ellipsoids, perturbed 3-discs and rectangles. We introduced a degree of randomness to these shapes, altering the positions of their constituent points to simulate natural variations. For each shape, we generated 20 differently-sized point clouds, consisting of 400 to 900 points, and saved them in `m_shapes_data.pkl`.

Our first step involves standardizing the data. We achieved this by confining each shape within a uniformly sized box. Simultaneously, we translated each shape to align its center with the origin $(0, 0, 0)$. This process ensured that all shapes are comparable in size and positioning. File `standardize.py` includes function for standardization of a point cloud. We applied this function to our point clouds and saved the output in `scaled_centered_shapes_data.pkl`.

Second step was building Rips Complexes for each point cloud. We initiated the analysis by expanding spheres (or balls) around each point in the point clouds, progressively increasing their radii. This method allowed us to construct a sequence of Rips complexes. From these Rips complexes, we computed persistence diagrams. These diagrams capture the essential topological features of each shape, highlighting their birth and death within the evolving complex. We transformed these persistence diagrams into a numerical format suitable for machine learning models. For this, we used persistence images, a technique that effectively vectorizes the topological data. As mentioned in the beginning, to enhance the distinction between shapes that might have similar persistence diagrams (like a line segment and a rectangle), we created a modified set of shapes. We did this by removing a small open ball from the center of each original shape and recalculating their persistence diagrams and images. We later found out that we get the best results if we do this twice: one time we remove ball with a bigger radius and one time with a smaller one. File `remove_ball.py` includes function that removes a ball with specified radius from the origin (which is now also a center of standardized point clouds). We used this function on standardized point clouds and saved the output in `modified_shape_data.pkl`.

We combined the vectorized data from both the original and modified shapes, creating a feature set for each point cloud. Following a train-test split, we employed a Support Vector Machine (SVM) model for classification. The model was optimized and then evaluated on the test set to assess its performance in distinguishing between the various shapes.

3 4-dimensional objects

We modified our model to recognize the shapes in 4-dimensional space. With script `m_create_4d_point_clouds.py`, we create 20 point clouds for each of the folowing 4-dimensional shapes: sphere, line segment, ellipsoid, 4-disc and perturbed 4-disc.