UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Low-Rank Approximation and Diffusion Maps

Máster Universitario en Investigación e Innovación en TIC (i^2-TIC)

Autor: DORADO ALFARO, Sara

Tutor: DORRONSORO IBERO, José Ramón

Cotutor: FERNÁNDEZ PASCUAL, Ángela

Departamento de Ingeniería Informática

September 4, 2018

Contents

List of Figures

List of Tables

List of Algorithms

Resumen

La teoría de la reducción de la dimensionalidad es fundamental para muchos problemas de Aprendizaje Automático. Existen multitud de enfoques, pero este trabajo se centrará en los métodos de aprendizaje de variedades. El punto de partida es asumir que los datos viven en una variedad de dimensión menor que la de partida para lograr entender el fenómeno subyacente que los ha generado. Dentro de este campo, es de especial interés, debido a su fuerte base matemática, el algoritmo conocido como Mapas de Difusión, objeto principal de este trabajo.

Primero realizaremos un estudio de los Mapas de Difusión así como de la teoría matemática necesaria para su correcta comprensión, estudiando conceptos como los Grafos de Semejanza y sus Laplacianos y la Distancia de Difusión. El principal inconveniente de los Mapas de Difusión, así como de otros algoritmos espectrales, es que requiere la diagonalización de una matriz cuadrada cuya dimensión es el número de ejemplos. Por lo tanto, su coste computacional es $\mathcal{O}(N^3)$, donde N se refiere al número de ejemplos. Es por ello que uno de los objetivos de este trabajo es calcular una aproximación de rango bajo para los Mapas de Difusión mediante el método de Nyström. Además, para evaluar la calidad de la aproximación, propondremos una métrica que se basa en el error de reconstrucción de la matriz de difusión. Por otro lado, existe otro problema cuando se quiere dar la proyección de un ejemplo que no está en la muestra inicial utilizada para el cálculo de la transformación. Es necesario rehacer el análisis espectral de la matriz, lo que es especialmente crítico si las aplicaciones tienen restricciones de funcionamiento en tiempo real. En este trabajo también analizaremos dos propuestas para paliar este coste: aprender el mapeo por medio de redes neuronales para regresión (Redes de Difusión), pudiendo así calcular la proyección para un nuevo ejemplo, y calcular una extensión de la transformación para un nuevo punto con el método de Nyström.

A la hora de mostrar los resultados se van a utilizar tres conjuntos de datos, uno de los cuales será sintético. Para todos ellos, se calculará la transformación a un espacio de menor dimensión por medio de Mapas de Difusión con el objetivo de extender los mismos a ejemplos fuera de muestra y evaluar la aproximación de rango bajo conseguida por el método de Nyström. Además, se calcularán extensiones para patrones fuera de muestra y se compararán los resultados obtenidos, tanto por las Redes de Difusión como por el método de Nyström, de forma visual.

En resumen, en cuanto a la calidad de la aproximación de rango bajo veremos que, como cabría esperar, incrementar el número de ejemplos en el conjunto de entrenamiento conlleva una reducción del error de reconstrucción de la matriz de difusión. En cuanto al funcionamiento de los métodos para extender el mapeo a ejemplos fuera de muestra, observaremos que tanto el método de Nyström como las Redes de Difusión obtienen resultados visualmente similares, proyectando ejemplos de la misma clase en las mismas regiones del espacio.

Este trabajo da lugar a nuevas líneas de investigación, ya que como trabajo futuro es de especial interés, entre otros, conseguir comparar la calidad de las extensiones para ejemplos fuera de muestra.

Abstract

The theory of dimensionality reduction is fundamental for many problems of Machine Learning. There are many approaches, but this work will focus on the methods of Manifold Learning. The starting point is to assume that data live in a manifold of smaller dimension than the starting one in order to understand the underlying phenomenon that has generated them. Within this field, it is of special interest, due to its strong mathematical foundation, the algorithm known as Diffusion Maps, the main object of this work.

First we will study the theory of Diffusion Maps, as well as the mathematical theory necessary for its correct understanding, studying concepts such as Similarity Graphs and their Laplacians, and the Diffusion Distance. The main drawback of Diffusion Maps, as well as other spectral algorithms, is that they require the diagonalization of a square matrix whose dimension is the number of examples. Therefore, its computational cost is $\mathcal{O}(N^3)$, where N refers to the number of examples. Because of that, one of the main objectives of this work is to compute a low-rank approximation for Diffusion Maps using Nyström's method. In addition, in order to evaluate the quality of the approach, we will propose a metric that is based on the reconstruction error of the diffusion matrix. On the other hand, there is another problem when giving the embedding for examples that were not in the initial sample used to compute the embedding. It is necessary to redo the spectral analysis of the matrix, which is especially critical if the applications have operating restrictions in real time. In this work we will also analyze two proposals to allevaite this cost: to learn the embedding by means of neural networks for regression (Diffusion Nets), being able to compute the embedding for a new example, and to compute an extension of the embedding with Nyström's method.

Regarding the results, three datasets will be used, one of which will be synthetic. For all of them, we will compute the embedding via Diffusion Maps with the objective of extending it to out-of-sample (OOS) examples and to evaluate the low range approximation achieved by Nyström's method. In addition, extensions for OOS patterns will be computed via Diffusion Nets and Nyström's method.

To sum up, regarding the low-rank approximation quality we will see that increasing the number of examples in the training set entails a reduction in the reconstruction error of the diffusion matrix, as we could expect. Regarding OOS extensions, we will observe that both, Nyström's method and Diffusion Nets, obtain visually similar results, embedding examples of the same class in the same regions of space.

This work gives rise to new lines of research, since as future work it is of special interest, among others, to be able to compare the quality of the extensions for OOS examples.

Acknowledgements

En primer lugar quiero agradecer a José Ramón Dorronsoro Ibero y a Ángela Fernández Pascual haber sido unos tutores excepcionales. Gracias por los conocimientos, la comprensión y el tiempo dedicado. Nada de esto habría sido posible sin vosotros. Gracias también a Carlos María Alaiz Gudín y a Alejandro Catalina Feliú por haberme acompañado en algunas de las reuniones semanales, teniendo paciencia para escucharme y corregirme.

Gracias a la Cátedra UAM-IIC de Ciencia de Datos y Aprendizaje Automático por la ayuda para máster concedida, que me ha permitido dedicarme a tiempo completo a mis estudios. Gracias al equipo docente de la Escuela Politécnica Superior y de la Facultad de Ciencias por haberme enseñado todo lo posible, siempre estando dispuestos a concertar tutorías y a aportar ideas.

Un enorme gracias a mi familia. Gracias mamá, por ayudarme cada día y por esforzarte tanto. Gracias papá, por todos los consejos y por enseñarme que tenemos seis sentidos, incluyendo el sentido común. Y gracias a mi hermano Sergio, por las charlas de futuro y los ratos juntos. Son lo mejor de cada día. Gracias abuelos, tíos y primos por vuestro apoyo y cariño incondicional.

A todos mis compañeros de aventuras estos últimos años. A pesar de los momentos duros, me llevo grandes recuerdos y amigos. En especial, gracias a Carlos por haberme acompañado y ayudado este último año. Y gracias a mis compañeras de deporte, las chicas del Sanfer Senior, porque me habéis dado una vía de escape siempre que la he necesitado. Teneros es genial.

Chapter 1

Introduction

1.1 Motivation

In the context of Artificial Intelligence and in the era of Big Data, dimensionality reduction plays a central role. By reducing the number of variables or features of a given problem we do not only achieve more efficiency, but many times we obtain a higher degree of accuracy. Methods of dimensionality reduction, both in classification and regression problems, try to change the representation of the original data by embedding it into a space of lower dimension in which a large part of the data information is retained.

In this work, we will study methods that suppose data lie in a manifold, a field known as manifold learning [2]. These methods have several applications, starting from data visualization to clustering [3]. Because of their good performance, spectral algorithms are very popular among the existing literature. On the one hand, we have the Laplacian Eigenmaps algorithm [4], which gives foundation to the algorithm of Spectral Clustering [5] and has been exhaustively studied. On the other hand, we find a more general technique, which are Diffusion Maps [6], the method that this work focuses on. This algorithm is specially interesting due to its strong mathematical foundation, which lies on the fields of similarity graphs and random walks. Diffusion Maps have a clear advantage over other manifold learning methods: they embed the data into a space in which the standard distance, that is the euclidean distance, approximates the Diffusion Distance in the original space, a robust measure.

The main disadvantage of Diffusion Maps is its computational complexity. In order to give the embedding, we need to compute the eigenvalues and eigenvectors of a square matrix of dimension the number of examples. The first problem of this is that we may not be able to compute the Diffusion Map of a huge dataset. The second problem comes when we want to compute the embedding of new examples, which are known as out-of-sample examples. With the arrival of a new pattern, it is necessary to recalculate the eigenvectors and eigenvalues of a similarity matrix, whose size is, again, the number of examples.

Aiming to solve these problems, we can find several proposals in the literature. One of them is trying to compute a low-rank approximation for the Diffusion Map [4]. That is, to compute the eigenvalues and eigenvectors of the similarity matrix of a small sample and, then, try to extend it to other patterns, which are the out-of-sample patterns. In this

sense, we can incorporate Nyström's method [7] to Diffusion Maps.

Other approach would be to compute the embedding for a given subset and learn it via any Machine Learning method. This could be, for example, neural networks for regression (the embedding is continuous) in which the target is the embedding we want to extend. This was introduced in [1] under the name of Diffusion Nets.

Thus, the motivation of this work is to review the current state-of-the-art of Diffusion Maps and present the mathematical theory necessary to reach a good understanding of this field. Apart from this review, we will also experiment with different datasets to measure the low-rank approximation quality reached by Nyström's encoding when applied to Diffusion Maps. In addition, we will compare the results obtained when trying to extend the embedding to out-of-sample examples via any of the previously mentioned approaches: Nyström's encoding and Diffusion Nets.

1.2 Objectives

The main objective of this work is to introduce a research line focused around low-rank approximation and out-of-sample extension in Diffusion Maps. With this general purpose, this work is focused on the following topics:

- To study and understand the mathematical background of Diffusion Maps.
- To study the theory of Similarity Graphs and Laplacians.
- To study the Nyström's method and its integration with Diffusion Maps.
- To study Diffusion Nets.
- To implement these concepts and methods and test them in synthetic and real datasets, comparing its performance to extract some conclusions.

1.3 Structure

In this context, this work is structured around three main chapters:

1. **Manifold learning and Difussion Maps**, where we introduce and develop the necessary mathematical background and the algorithm of Diffusion Maps. We introduce Principal Component Analysis, giving an example of the necessity of non-linear methods for dimensionality reduction. This chapter also contains the theory of Similarity Graphs and Laplacians and the algorithm of Laplacian Eigenmaps.
2. **Out-of-sample extension**, where different approaches to extend the embedding computed by Diffusion Maps are explained. Firstly, we introduce the Nyström's approach and its application to Diffusion Maps. We will also give an idea to measure the low-rank approximation quality achieved by the Nyström's method. Secondly, the algorithm of Diffusion Nets is also explained, introducing its advantages and applications.

3. **Experiments**, where we will implement and compare these methods. We will work on three different problems: a synthetic three-dimensional helix, the Red Wine dataset, which contains examples of wine quality, and the Vowel dataset, for recognition of the eleven steady state vowels of British English. For each dataset, we will compute an embedding via Diffusion Maps that will be extended both by the Nyström's encoding, measuring the low-rank approximation quality, and by Diffusion Nets.

Finally, we include a chapter of conclusions and further work, where we explore future lines of research.

Chapter 2

Manifold Learning and Diffusion Maps

In this chapter we will cover a set of mathematical concepts that will be essential to develop the analysis of the models and problems that we will deal with in the third section of this work. In this document the initial dataset, which is composed of N patterns or examples, is denoted as $\mathcal{D} = \{x_n\}_{n=1}^{N}$, with $x_n \in \mathbb{R}^m$. We will say m is the original dimension. For the most part, the presented methods assume data lie on a low-dimensional manifold.

Given a problem, methods for dimensionality reduction aim to find a new space, which is usually called *embedding space* or *projection space*, where data are largely explained in some sense and maintain initial relations, such as local distances. Some methods, like PCA [8, 2], aim to find a projection where variables are uncorrelated and the variance retained is maximum. However, these methods are linear and, hence, they have limitations. In this work we will focus on methods that try to find meaningful structures in datasets, a paradigm known as manifold learning. This problem, which is slightly different to dimensionality reduction, aims to gain insight and understanding of the process that generated the data. The key is to assume that data live in a manifold of lower dimension than the original m-dimensional space.

It may be useful to recall the concept of manifold. Informally, a *manifold* is a topological space that locally resembles an Euclidean space and may have a more complicated global structure. It is locally considered as the image of a domain with lower dimension. In this sense, manifold learning methods ideally have the objective of finding an application, Ψ, from the original space to the projection space, such that

$$\begin{aligned} \Psi : \mathcal{D} \subset \mathbb{R}^m &\to Y \subset \mathbb{R}^d \quad (d < m) \\ x &\mapsto \Psi(x) = y \ , \end{aligned}$$

where d denotes the dimension of the space in which we are projecting the data. It is called the *embedding dimension*.

This chapter is organized as follows:

1. In Section 2.1 we will see a derivation of the Principal Component Analysis (PCA) algorithm, which is a linear method for dimensionality reduction. We will also see

an example to motivate non-linear methods.

2. Section 2.2 will introduce the concept of similarity graph. We will also cover some useful theorems and applications of this mathematical theory that will lead to the next section. The Laplacian Eigenmaps algorithm is also introduced in Section 2.2.2.

3. In Section 2.3 we will see in detail, including some implementational aspects, the algorithm known as Diffusion Maps.

4. In Section 2.4 we will introduce the Spectral Clustering algorithm, which has become one of the most popular modern clustering algorithms.

2.1 Principal Component Analysis: PCA

Principal Component Analysis [8, 2] (PCA) is a dimensionality reduction algorithm whose aim is to find a subspace in which the retained variance of the projected data is maximum. As we shall see, this method is based on the spectral analysis of the covariance matrix of the initial dataset $\mathcal{D} = \{x_1, \dots, x_N\}$. There are two approaches to derive this algorithm, as PCA can be seen from a *maximum variance formulation* or *minimum error formulation*. Both approaches can be found in [8], but here we will focus on the first one since it has some similarities with other methods that will be explained in this work.

The objective is to project $\mathcal{D}$ into a d dimensional space, where $d < m$. Suppose $d = 1$ and let $u \in \mathbb{R}^m$ be the direction in which we want to project the data. In order to avoid infinite solutions it is necessary to add a restriction over the vector norm, which will be required to be unitary. That is $u^T u = 1$. Since u is unitary the projection of a given point $x_n \in \mathcal{D}$, $n = 1, \dots, N$, over the direction of u can be expressed as

$$y_n = u^T x_n \ .$$

The average of the projection can also be expressed in terms of u as

$$\overline{y} = u^T \overline{x} \ ,$$

where $\overline{x}$ denotes the average of the original dataset, that is

$$\overline{x} = \frac{1}{N} \sum_{n=1}^{N} x_n \ .$$

Finally, the variance of the projection is

$$\mathrm{Var}[y] = \frac{1}{N} \sum_{n=1}^{N} (y_n - \overline{y})^2 = u^T S u \ ,$$

where

$$S = \frac{1}{N} \sum_{n=1}^{N} (x_n - \overline{x})(x_n - \overline{x})^T$$

is the covariance matrix of the initial dataset. The optimization problem would be

$$\begin{aligned} \max_{u \in \mathbb{R}^m} \quad & u^T S u \\ \text{s.t.} \quad & u^T u = 1, \end{aligned} \tag{2.1.1}$$

whose maximum of the problem can be found using the Lagrangian associated to the Problem 2.1.1.

$$\mathcal{L}(u, \lambda) = u^T S u - \lambda(u^T u - 1) \ ,$$

where λ is a Lagrange multiplier. Taking the gradient with respect to u and equalling to zero we obtain

$$Su - \lambda u = 0 \iff Su = \lambda u \ .$$

That is to say, the stationary solutions of the Lagrangian are the eigenvectors of S and the Lagrange multipliers are the corresponding eigenvalues. Since S is a covariance matrix, it is positive semi-definite: it is easy to see that for any $z \in \mathbb{R}^m$ $z^T S z \geq 0$, as we have

$$\begin{aligned} z^T S z^T &= z^T \left(\frac{1}{N} \sum_{n=1}^{N} (x_n - \overline{x})(x_n - \overline{x})^T \right) z \\ &= \frac{1}{N} \sum_{n=1}^{N} z^T (x_n - \overline{x})(x_n - \overline{x})^T z \\ &= \frac{1}{N} \sum_{n=1}^{N} \left\| (x_n - \overline{x})^T z \right\|^2 \geq 0 \ . \end{aligned}$$

In addition, S is symmetric, so it can be diagonalized into an orthonormal basis (see Theorem 1, Section 2.2.1 of this work). Hence, if we evaluate the objective function in a solution (u^*, λ^*), we obtain

$$(u^*)^T S u^* = (u^*)^T \lambda^* u^* = \lambda^* \ .$$

Thus, we can conclude that the variance is maximized in the direction of the eigenvector associated with the largest eigenvalue. From here the definition is incremental: as we have already explored the direction u, we are interested on the remaining orthonormal space. In this space, the variance is maximzed in the direction v, the eigenvector associated with the second largest eigenvalue of S, which is orthonormal to u, and so on.

An important observation about PCA is that, if we set $d = m$, i.e. the target dimension equal to the original dimension, we will find a projection in which the variables are uncorrelated, so the covariance matrix is diagonal. If the data is normalized, then the covariance will be the identity matrix. In addition, as we have said, S can be diagonalized in an orthonormal basis, so we can write $S = PDP^T$, where P is the matrix with the eigenvectors of S in columns and D is a diagonal matrix containing the eigenvalues. The projection of x in the basis composed by the eigenvectors of S can be written as $\alpha = P^T x$. Thus,

$$\mathrm{Var}(\alpha) = \mathrm{Var}(P^T x) = P^T S P = D \ .$$

So the components are uncorrelated and their variance is the corresponding eigenvalue.

Even though PCA is a linear method, it is widely used. However, when applying PCA for dimensionality reduction we encounter some limitations. Figure 2.1.1 contains the 2-dimensional embedding computed via PCA of a 3-dimensional helix that we will explain with detail in further sections. Unfortunately, points that are far away in the original space are near in the projection space. Observe, for example, the orange and the greenish blue points, which appear separated in the original space and overlap in the embedding space, which is not desirable.

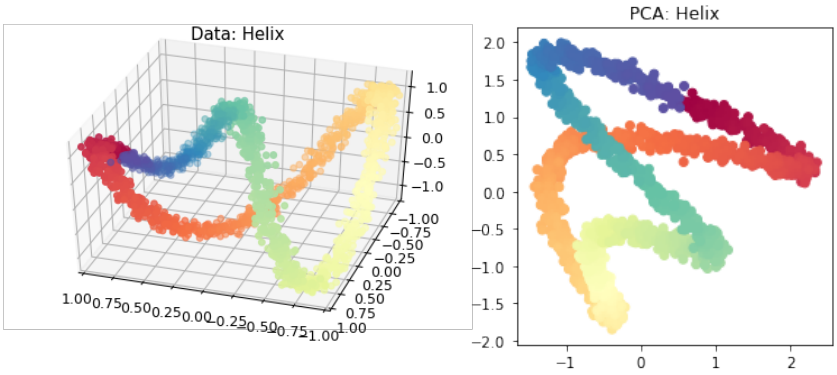

Figure 2.1.1: Helix dataset. Left: data in the original space. Right: embedding computed via PCA.

2.2 Similarity Graphs and Laplacians

In this section we will introduce some mathematical definitions and results that will give the foundation of the Laplacian Eigenmaps and Spectral Clustering algorithms [5, 9, 3], that often outperform traditional clustering algorithms, such as k-means or single linkage [10]. Given a set of points, $\mathcal{D} = \{x_1, \dots, x_N\}$, and a measure of similarity, w_{ij}, between x_i and x_j we can construct an undirected graph $G = (\mathcal{X}, E)$, in which the vertices, $\mathcal{X}$, are the set of points and the set of edges, E, is weighted by w_{ij}. Weights are commonly obtained by means of a positive definite and symmetric kernel $k(x, y)$. Given such a graph, we can define its adjacency matrix.

Definition 1 (Adjacency matrix)**.** *The adjacency matrix of the graph G is $W = (w_{ij})$, with $i, j = 1, \dots, N$. As G is undirected, it is required that $w_{ij} = w_{ji}$.*

The adjacency matrix will be also called similarity matrix, as it contains the similarity between points. Note that $w_{ij} = 0$ implies that x_i and x_j are not directly connected in the graph. This is not the standard in graph theory, as normally the edges are weighted by the cost of traveling between nodes, which is typically a decreasing function of the similarity. Thus, generally, the weight between two nodes that are not connected should be infinite. However, Similarity Graphs contain the similarity between vertices, which should be 0 if two points are not connected in the graph, i.e. the two points are not considered to be similar.

Definition 2 (Degree of a vertex)**.** *The degree of the vertex x_i is computed as*

$$d(x_i) = d_i = \sum_{j=1}^{N} w_{ij} \ .$$

Definition 3 (Degree matrix)**.** *The degree matrix of G is $D = (d_{ij})$, such that*

$$d_{ij} = \begin{cases} d_i \text{ if } i = j \\ 0 \text{ otherwise} \ . \end{cases} \tag{2.2.1}$$

When working with subgraphs composed by a subset of vertices, A, and the edges of the graph restricted to those vertices, E_A, we will write $(A, E_A) \subset (\mathcal{X}, E)$. The complementary

of A will be denoted as $\bar{A}$ and the number of vertices in A is written as $|A|$. We will say $A \subset \mathcal{X}$ is connected if every pair of vertices can be connected by a path totally contained in E_A. In addition, if there are no connections between A and $\bar{A}$, we will say A is a connected component. Thus, we will say G is connected if it only contains one connected component.

Definition 4 (Indicator vector)**.** *The indicator vector of $A \subset \mathcal{X}$, $\mathbb{1}_A$, is*

$$\mathbb{1}_A = (f_1, \ldots, f_N)^T ,$$

where $f_i = 1$ if $x_i \in A$ and 0 otherwise.

Given a dataset, $\mathcal{D}$, there are several popular constructions that can be used to transform the data into a graph. The most used approaches are given in [5] and they are the following.

The simplest approach is the **ϵ-neighborhood graph**, in which two vertices are connected when the distance between them is less than a given parameter ϵ. As the similarity between the connected points is in the same scale (at most ϵ), this graph is usually considered as an unweighted graph. Simply, we set $w_{ij} = 1$ if x_i and x_j are connected and 0 otherwise.

We also have the k **nearest neighbor graph**, in which each vertex is connected with its k nearest neighbors. This approach leads to a directed graph. Depending on the ways of making the graph undirected, we can distinguish:

- The vertices x_i and x_j are connected if x_i is among the k nearest neighbors of x_j **or** vice versa.
- The vertices x_i and x_j are connected if x_i is among the k nearest neighbors of x_j **and** vice versa. This one is known as mutual k nearest neighbor graph.

The first approach creates a denser graph, in the sense that it contains more edges than the second one. In this case, in contrast to the ϵ-neighborhood graph, the similarity w_{ij} between x_i and x_j is set to 1 if x_i and x_j are connected and to 0 otherwise.

Finally, in the **totally connected graph** all vertices of G are connected by means of a kernel that represents the notion of similarity. The Gaussian kernel is commonly used, leading to the similarity

$$w_{ij} = k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) ,$$

where σ is a parameter that controls the width of the neighborhoods and $x_i, x_j \in \mathcal{X}$. Typically, the equation of this kernel is written as

$$w_{ij} = \exp\left(-\gamma \|x_i - x_j\|^2\right) ,$$

where $\gamma = \frac{1}{2\sigma^2}$. However, in this work we will the σ-notation, since it will be more comfortable when performing the experiments. Note that any symmetric and positive definite kernel can be used instead. This is the type of graph usually built in Diffusion Maps.

2.2.1 Graph Laplacians and their Basic Properties

The main tools for Spectral Clustering are graph Laplacian matrices. There exists a whole field dedicated to the study of those matrices, called Spectral Graph theory. Given a dataset $\mathcal{D}$, its associated graph G, the adjacency or similarity matrix W and the degree matrix D, the graph Laplacian can be defined in two ways depending on the normalization: the Unnormalized and the Normalized graph Laplacian. For this section, it may be useful to recall the Spectral Theorem.

Theorem 1. *(Spectral theorem) Let A be a symmetric matrix of dimension $N \times N$. Then, it can be diagonalized with an orthonormal basis, i.e., $A = U\Lambda U^T$, where*

- *Λ is a diagonal matrix with real entries, and*
- *the columns of U form an orthonormal basis of $\mathbb{R}^N$.*

In the following, we do not necessarily assume that the eigenvectors are normalized so the vector v and $a \times v$, $a \in \mathbb{R}$, are considered to be the same eigenvector. Eigenvalues will always be ordered from lowest to highest, respecting multiplicities. By *the first k eigenvectors* we refer to the eigenvectors corresponding to the k smallest eigenvalues.

Definition 5 (Unnormalized graph Laplacian)**.** *The Unnormalized graph Laplacian of a graph is defined as*

$$L = D - W \ . \tag{2.2.2}$$

Note that $W = (w_{ij})$ is a similarity matrix, so we have $w_{ij} \geq 0$ for $i, j = 1, \ldots, N$. Some important facts of the Unnormalized graph Laplacian are summarized in the following theorem, whose proof can be found in [5].

Theorem 2. *L satisfies the following properties:*

1. *$\forall f \in \mathbb{R}^N, f'Lf = \frac{1}{2} \sum_{i=1,j=1}^{N} w_{ij}(f_i - f_j)^2$.*
2. *L is symmetric and positive semidefinite.*
3. *The smallest eigenvalue of L is 0. The corresponding eigenvector is the constant vector $\mathbb{1} \in \mathbb{R}^N$. The normalized eigenvector would be $\frac{\mathbb{1}}{\sqrt{N}} \in \mathbb{R}^N$.*
4. *L has N non-negative real eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N$ and real eigenvectors.*

There is another matrix which is called Normalized graph Laplacian in the literature and it is widely used when doing Spectral Clustering.

Definition 6 (Normalized graph Laplacian)**.** *We call the Normalized graph Laplacian of a similarity graph G to the matrix*

$$L_{rw} = D^{-1}L = I - D^{-1}W \ .$$

The reason to write L_{rw} is that this Laplacian is related to the probability matrix of a random walk defined over the graph. We will see this in more detail in Section 2.3. The

Normalized graph Laplacian is closely related to the Unnormalized version. In fact, it is easy to check that λ is an eigenvalue of L_{rw} with eigenvector v if and only if λ solves the generalized eigenproblem

$$Lv = \lambda Dv \ . \tag{2.2.3}$$

That is to say, like L, L_{rw} is positive definite and has N non-negative eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N$ and real left eigenvectors $v_1, \ldots, v_N$. The main result for the Normalized graph Laplacian is related to the number of connected components in the graph, as detailed in Theorem 3. The proof can be found in [5].

Theorem 3. *Let $G = (\mathcal{X}, E)$ be an undirected graph with non-negative weights. Let $A_1, \ldots, A_k$ be its connected components. Then, the multiplicity of the eigenvalue 0 in L_{rw} is k. The corresponding eigenvectors are $\{\mathbb{1}_{A_1}, \ldots, \mathbb{1}_{A_k}\}$.*

2.2.2 Spectral Dimensionality Reduction and Laplacian Eigenmaps

The Laplacian Eigenmaps (LE) method is a very successful procedure for dimensionality reduction [2] while preserving local properties under certain conditions, as we will see below. In the following we will suppose a fully connected graph. Given a dataset $\mathcal{D} = \{x_i\}_{i=1,\ldots,N}$ and a similarity measure between points, $w_{ij} = k(x_i, x_j)$, the algorithm works as follows:

1. Build the undirected similarity graph $G = (\mathcal{X} = \mathcal{D}, E)$ and compute the similarity matrix W following any of the approaches shown in Section 2.2.
2. Solve the problem $Lv = \lambda Dv$, where L and D are defined in Equations (2.2.2) and (2.2.1) respectively.
3. Compute the embedding:

   $$\Psi : x_i \in \mathcal{D} \to (v_2(x_i), \ldots, v_d(x_i)) \ ,$$

   where $v_k(x_i)$ denotes the i-th component of the k-th generalized eigenvector from Equation (2.2.3). Note that just $v_1(x)$ is constant as we are supposing a fully connected graph.

The justification of this algorithm can be found in [2]. If we denote the embedding matrix by $Y \in \mathbb{R}^{N \times d}$, a reasonable objective function would be to penalize points that are close in the original space (and hence the edge between them has a large weight) but are mapped far apart in the embedding space. Therefore, a plausible objective function is

$$\sum_{i,j} \|y_i - y_j\|^2 w_{ij} \ ,$$

where y_k denotes the embedding of the point x_k, that is to say, the k-th row of Y. It can be shown that

$$\sum_{i,j} \|y_i - y_j\|^2 w_{ij} = \operatorname{trace}(Y^T L Y) \ .$$

Hence, we get the following optimization problem:

$$\underset{Y^T D Y = \mathcal{I}}{\operatorname{argmin}} \operatorname{trace}(Y^T L Y) \ ,$$

where $\mathcal{I}$ denotes the identity matrix. We constrained one degree of freedom in order to avoid an arbitrary solution. Since L is positive definite (see Theorem 2), this optimization problem is solved by the eigenvectors corresponding to the first d eigenvalues of the generalized problem

$$Lv = \lambda Dv \ , \tag{2.2.4}$$

which will be denoted as v_i, $i = 1, \ldots, d$. That is to say, the solution is the matrix

$$Y = [v_1, \ldots, v_d] \ ,$$

where v_j, $j = 1, \ldots, d$, denotes the j-th solution of the Equation (2.2.4). Thus, the columns of Y are the first d left eigenvectors of L_{rw}.

As we will see in the next section, Laplacian Eigenmaps, which are a particular case of Diffusion Maps, handle only manifolds from which the data is sampled uniformly, something that rarely happens in real Machine Learning tasks. Diffusion Maps address this problem.

2.3 Diffusion Maps

Diffusion Maps [6] are another technique for finding meaningful geometric descriptions for datasets even when the observed samples are non-uniformly distributed. Similarity graphs seen in Section 2.2 give us a powerful tool to depict the notion of geometry. In this algorithm, Coifman and Lafon provide a new motivation for Normalized graph Laplacians by relating them to Diffusion Distances.

In the original work the algorithm is motivated with continuous data, so we work in a probability space $(\mathcal{X}, \mathcal{A}, \mu)$ where a symmetric kernel that verifies $k(x, y) \geq 0$ presents the notion of local structure (similarity). The degree of a vertex $x \in \mathcal{X}$ is the integral

$$d(x) = \int_{\mathcal{X}} k(x, y) d\mu(y) \ .$$

So, we define the transition probability between vertices as

$$p(x, y) = \frac{k(x, y)}{d(x)} \ .$$

Even though the kernel $p(x, y)$ is not symmetric, it inherits the positive-preserving property of $k(x, y)$. That is, $p(x, y) \geq 0$. In addition, we have gained the conservation property

$$\int_{\mathcal{X}} p(x, y) d\mu(y) = 1 \ .$$

This means that p can be viewed as the transition kernel of a Markov chain on $\mathcal{X}$. The diffusion operator associated to $p(x, y)$ is defined as

$$Pf(x) = \int_{\mathcal{X}} p(x, y) f(y) d\mu(y) \ .$$

We say $\psi(x)$ is an eigenfunction, with associated eigenvalue λ, of the operator P if it verifies

$$P\psi(x) = \lambda\psi(x) \ .$$

In practice, we always work with finite samples. The diffusion operator is then a matrix $P_{ij} = p(x_i, x_j)$ and $p(x_i, x_j)$ is now the transition probability in a random walk defined over the initial dataset $\mathcal{X} = \{x_1, \ldots, x_N\}$. In the same way, instead of working with eigenfunctions we work with eigenvectors. The transition matrix P is related to L_{rw} from Definition 6 as

$$L_{rw} = I - D^{-1}W = I - P \ ,$$

where W is the similarity matrix from Definition 1. That is, $W_{ij} = k(x_i, x_j)$, $x_i, x_j \in \mathcal{X}$. From this we can conclude that if λ is an eigenvalue with right eigenvector v of L_{rw}, then $1 - \lambda$ is an eigenvalue with right eigenvector v of P. As a consequence of Theorem 3 we can deduce that the multiplicity of the eigenvalue $\lambda = 1$ in P is the number of connected components. In addition, if we suppose a connected graph, as P defines the transition matrix of a random walk, we can assert that P has a discrete sequence of eigenvalues

$$1 = \lambda_0 > \lambda_1 \geq \lambda_2 \geq \ldots \geq 0 \ ,$$

and, in addition, $P\psi_l = \lambda_l \psi_l$, where ψ_l denotes the l-th eigenvector of P, which are real eigenvectors. As already mentioned, the eigenvector associated to $\lambda_0 = 1$ is constant, collapsing all the elements of each component around 1 and, therefore, not providing any information. In [6] the authors gave a measure, known as Diffusion Distance, that establishes how two points are connected in a graph.

Definition 7 (Diffusion Distance)**.** *A family of Diffusion Distances $\{D_t\}_{t\in\mathbb{N}}$ in time t is defined as:*

$$\begin{aligned} D_t^2(x, y) &= \|p_t(x, \cdot) - p_t(y, \cdot)\|^2_{L^2(\mathcal{X}, d\mu)} \\ &= \int_{\mathcal{X}} (p_t(x, u) - p_t(y, u))^2 d\mu(u) \ , \end{aligned}$$

where $p_t(x, y)$ denotes the probability of traveling from x to y in time t.

Note that if we have $D_t(x, y) \simeq 0$ then we also have $p_t(x, u) \simeq p_t(y, u)$ for $u \in \mathcal{X}$. This means that it should be also posible to travel from x to y in time t. In practice, the integrals are intractable. Given a dataset $\mathcal{D}$, the discrete version of the Diffusion Distance for $x, y \in \mathcal{D}$ is

$$\begin{aligned} D_t^2(x, y) &= \|p_t(x, z) - p_t(y, z)\|_d^2 \\ &= \sum_{z\in\mathcal{D}} (p_t(x, z) - p_t(y, z))^2 d(z) \ , \end{aligned}$$

where $p_t(x, y)$ is now the probability of traveling from x to y in t steps. The definition is intuitive: two points are closer the more short paths (with large weights) connect them. $D_t(x, y)$ involves all paths of length t. Hence, it is a robust measure. In addition, this measure can be expressed in terms of the eigenvalues and eigenvectors of P, as proved in [6].

Theorem 4. *We have*

$$D_t^2(x, y) = \sum_{l\geq 1} \lambda_l^{2t} \left(\psi_l(x) - \psi_l(y)\right)^2 \ ,$$

where $\psi_l(x)$ denotes the value of the l-th eigenvector of P. The case $l = 0$ is omitted because ψ_0 is constant, as we are assuming a fully connected graph.

We have seen that the eigenvalues of P are a decreasing sequence, so the diffusion distance can be approximated up to a certain precision $s(\delta, t)$, where δ is the relative accuracy, by truncating the summation,

$$D_t^2(x, y) \simeq \sum_{l=1}^{s(\delta,t)} \lambda_l^{2t}(\psi_l(x) - \psi_l(y))^2 \ .$$

Choosing δ is not trivial. In [6] the authors propose the following

$$s(\delta, t) = \max\{l \in \mathbb{N} : |\lambda_l|^t > \delta|\lambda_1|^t\} \ . \tag{2.3.1}$$

We will identify $s(\delta, t)$ with the dimension of the embedding, which we have denoted by d. In this way, we can formally define the concept of Diffusion Map.

Definition 8 (Diffusion Map)**.** *A Diffusion Map is a function $\Psi_t(x) : \mathcal{X} \to \mathbb{R}^d$ such that*

$$\Psi_t(x) = \begin{pmatrix} \lambda_1^t\psi_1(x) \\ \lambda_2^t\psi_2(x) \\ \vdots \\ \lambda_d^t\psi_d(x) \end{pmatrix} \ .$$

Theorem 5. *The diffusion map $\Psi_t : \mathcal{X} \to Y$ embeds the data into the Euclidean space $Y = \mathbb{R}^d$ in which the Euclidean distance is approximately the diffusion distance D_t in the original space.*

Proof. The Euclidean distance between two points, $y_1 = \Psi_t(x_1)$ and $y_2 = \Psi_t(x_2)$, in the embedding space is

$$\begin{aligned} \|\Psi_t(x_1) - \Psi_t(x_2)\|^2 &= \sum_{l=1}^{d} \left(\lambda_l^t\psi_l(x_1) - \lambda_l^t\psi_l(x_2)\right)^2 \\ &= \sum_{l=1}^{d} \lambda_l^{2t} \left(\psi_l(x_1) - \psi_l(x_2)\right)^2 \\ &\simeq D_t^2(x_1, x_2) \ . \end{aligned}$$

□

To sum up, the Diffusion Map is an embedding that projects the original data in a Euclidean space in which the Euclidean distance approximates the diffusion distance.

In this work we will try to extend the embedding given by Diffusion Maps to examples that were not in the initial dataset. The transition matrix $P = D^{-1}W$ is not symmetric, but when diagonalizing or giving an extension of the embedding for new patterns, it is convenient to define a symmetric kernel, whose eigenvectors compose an orthonormal basis of $\mathbb{R}^d$. Therefore, it is common to define a symmetric operator as

$$a(x_i, x_j) = \frac{k(x_i, x_j)}{\sqrt{d(x_i)}\sqrt{d(x_j)}} \ , \tag{2.3.2}$$

and, consequently, the matrix $A_{ij} = a(x_i, x_j)$, which can be expressed in terms of the kernel and degree matrices as $A = D^{-1/2}WD^{-1/2}$. Note that A is symmetric, so we are

not distinguishing between right and left eigenvectors anymore (they are the same but transposed). The eigenvalues and eigenvectors of A, which will be denoted as λ_l and ϕ_l respectively, are directly related to those of P (λ_l, ψ_l). We have

$$A\phi_l = \lambda_l \phi_l \ . \tag{2.3.3}$$

Multiplying both sides by $D^{-1/2}$, we obtain

$$PD^{-1/2}\phi_l = \lambda_l D^{-1/2}\phi_l \ .$$

So the eigendecomposition of P can be recovered by choosing, associated with the eigenvalue λ_l, the right eigenvector

$$\psi_l = D^{-1/2}\phi_l \ .$$

2.3.1 Anisotropic Diffusion

The density of the sample is not, in general, related to the geometry of the manifold. When the sampling on the manifold is not uniform, the diffusion maps may not recover the original geometry. We are interested in recovering the structure of the manifold regardless of the density of the sample, which suggests the following question: What is the influence of the density of the points and the geometry of the possible manifold underlying the data in the eigenvectors and the diffusion spectrum?

The *family of Anisotropic Diffusions* [6] introduces a new parameter, α, that can be tuned to specify the influence of the density of the sample points. The following is an outline of how the Anisotropic Diffusion works, emphasizing the similarities with the Diffusion Maps introduced in the previous section. Let $q(x)$ be the density of the points on the manifold.

1. Fix $\sigma \in \mathbb{R}$ and a kernel $k_\sigma = \exp\left(\frac{-\|x-y\|^2}{2\sigma^2}\right)$.

2. Define
$$q_\sigma(x) = \int_{\mathcal{X}} k_\sigma(x,y)q(y)dy \ ,$$
which is an approximation of the true density $q(x)$, and the kernel
$$k_\sigma^{(\alpha)}(x,y) = \frac{k_\sigma(x,y)}{q_\sigma^\alpha(x)q_\sigma^\alpha(y)} \ .$$
Setting $\alpha = 0$ we obtain $k_\sigma^{(\alpha)} = k_\sigma$.

3. Define the degree in terms of $k_\sigma^{(\alpha)}(x,y)$ as
$$d_\sigma^{(\alpha)}(x) = \int_{\mathcal{X}} k_\sigma^{(\alpha)}(x,y)q(y)dy \ ,$$
and define the anisotropic diffusion as
$$p_{\sigma,\alpha}(x,y) = \frac{k_\sigma^{(\alpha)}(x,y)}{d_\sigma^{(\alpha)}(x)} \ .$$

4. Apply Diffusion Maps as usual. The diffusion matrix is then

$$(P_{\sigma,\alpha})_{ij} = p_{\sigma,\alpha}(x_i, x_j) \ , i, j = 1, \ldots, N \ .$$

For a finite sample, the integrals are approximated by sums in the following way

$$\overline{q}_\sigma(x_i) = \sum_{j=1}^{N} k_\sigma(x_i, x_j) \ ,$$

$$\overline{d}_\sigma^{(\alpha)}(x_i) = \sum_{j=1}^{N} \frac{k_\sigma^{(\alpha)}(x_i, x_j)}{\overline{q}_\sigma^{\alpha}(x_i)\overline{q}_\sigma^{\alpha}(x_j)} \ ,$$

$$\overline{p}_{\sigma,\alpha}(x_i, x_j) = \frac{k_\sigma^{(\alpha)}(x_i, x_j)}{\overline{d}_\sigma^{(\alpha)}(x_i)} \ .$$

In order to understand the impact of the new parameter α in the algorithm we may introduce the Laplace-Beltrami operator of the manifold [11]. This operator is a generalization of the Laplace operator but applied over functions and it is related to the geometry of the manifold. We assume that the data, $\mathcal{X}$, is the entire manifold. If we define the operator

$$L_{\sigma,\alpha} = \frac{I - (P_{\sigma,\alpha})}{\sigma} \ ,$$

then, it is proved in [11] that for any function f, it is verified

$$\lim_{\sigma \to 0} L_{\sigma,\alpha} f = \frac{\Delta(f q^{1-\alpha})}{q^{1-\alpha}} - \frac{\Delta(q^{1-\alpha})}{q^{1-\alpha}} f \ , \tag{2.3.4}$$

where $\Delta(\cdot)$ denotes the Laplace-Beltrami operator. Some values of interest of α are discussed in [11]:

- When $\alpha = 0$ the diffusion is reduced to the classical problem with the normalized Laplacian L_{rw}. From the previous equation we obtain

  $$\lim_{\sigma \to 0} L_{\sigma,\alpha} f = \frac{\Delta(fq)}{q} - \frac{\Delta(q)}{q} f \ .$$

  That is to say, the density influence is very strong in this case. Assuming an uniform distribution in the manifold, i.e., that the points are equally distributed among the manifold, then the density is a constant. Thus, we can write $q = C$, where C is a real number. Hence, we have $\Delta(q) = 0$ and $\Delta(fq) = q\Delta(f)$. Thus we can simplify the expression to obtain

  $$\lim_{\sigma \to 0} L_{\sigma,\alpha} f = \Delta(f) \ .$$

  As we said before, assuming a constant density, the Laplace-Beltrami operator can be approximated by the graph Laplacian.

- When $\alpha = 1$, if the points are actually in a submanifold of $\mathbb{R}^d$, we get from Equation (2.3.4)

  $$\lim_{\sigma \to 0} L_{\sigma,\alpha} f = \Delta(f) \ ,$$

  so the geometry of the manifold is perfectly retrieved (in the limit). Thus, an approximation of the Laplace-Beltrami operator is always obtained independently of the density and the Riemmanian geometry of the dataset can be retrieved.

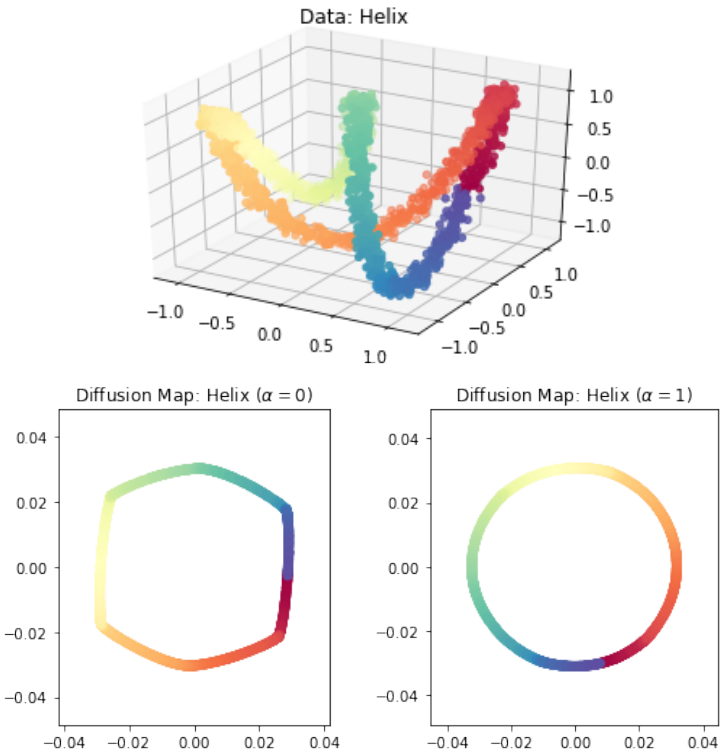

Figure 2.3.1: Influence of α on an helix example. Up: original curve. Down: the embedding via the graph Laplacian ($\alpha = 0$) and the embeddings via the Laplace–Beltrami approximation ($\alpha = 1$). In the latter case, the curve is perfectly unrolled

To give an idea, in Figure 2.3.1 we can see the original curve and the embedding of an helix with equation

$$\begin{aligned} x_i &= cos(\theta_i) \ , \\ y_i &= sen(2\theta_i) \ , \\ z_i &= sen(3\theta_i) \ , \end{aligned}$$

where $\theta_i \in (0, 2\pi)$; we added Gaussian white noise with standard deviation $\sigma_r = 0.5$. This example has been previously used in Section 2.1, where we saw that the 2-dimensional embedding computed via PCA was not able to retrieve the geometry of the manifold. In Figure 2.3.1, we can see that, when taking the density of the sample into account, the curve is perfectly unrolled. Note that when the density is not taken into account, spikes are observed in the embedding, corresponding to changes in the helix, where there is more density of points. When we apply $\alpha = 1$ this effect disappears, and the helix is embedded into a smooth 2-dimensional circumference.

2.4 Clustering

An important and recent application of Laplacian Eigenmaps and Diffusion Maps is known as Spectral Clustering. We have noted before that both algorithms try to embed the data into a space where local magnitudes are preserved. In the case of Diffusion Maps, the data

is projected into a Euclidean space where the usual notion of distance is similar to the Diffusion Distance D_t, which is a robust measure that takes into account probabilities of transition between points in the manifold. Recall that the quantity $D_t(x, y)$ involves all paths of length t that connect x to y and vice versa. As a consequence, this number is very robust to noise perturbation.

This section will also provide a practical example in which algorithms based in the euclidean distance, such as k-means in the original space, are not able to cluster the points perfectly. However, we will see that applying k-means in the embedding space for this particular dataset provides a perfect separation between classes.

2.4.1 Classic Algorithm: k-means

K-means [8] tries to find groups or clusters in the data. Firstly, K centroids are initialized, $\{\mu_1, \ldots, \mu_K\}$, and each pattern is then assigned to its closest centroid. The error is defined as

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \left\| x_n - \mu_k \right\|^2 \ , \tag{2.4.1}$$

where r_{nk} is 1 if the point x_n is associated with the k-th centroid. The optimization is done in two steps:

- First, r_{nk} is updated as

$$r_{nk} = \begin{cases} 1 & \text{if } k = \operatorname{argmin}_j \left\| x_n - \mu_j \right\|^2 \\ 0 & \text{otherwise .} \end{cases}$$

- Then we update the cetroids μ_k by minimizing Equation (2.4.1) with r_{nk} fixed.

Deriving (2.4.1) with respect to μ_k and equaling to zero, we obtain,

$$2 \sum_{n=1}^{N} r_{nk}(x_n - \mu_k) = 0 \ .$$

Therefore, solving the equation, we finally obtain

$$\mu_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}}.$$

That is to say, the optimal centroid of the cluster k is the average of the points in the cluster. These steps are repeated until the centroids stop being updated or a maximum number of iterations is reached. Note that this algorithm is sensible to the notion of distance. The Euclidean distance may not be the best selection, specially if we are working with data that lies in a manifold, which is the case studied in this work.

2.4.2 Spectral Clustering

To conclude this section, let us quickly sketch the application of Diffusion Maps to Spectral Clustering algorithms. Typically in the literature [5], Spectral Clustering algorithms are

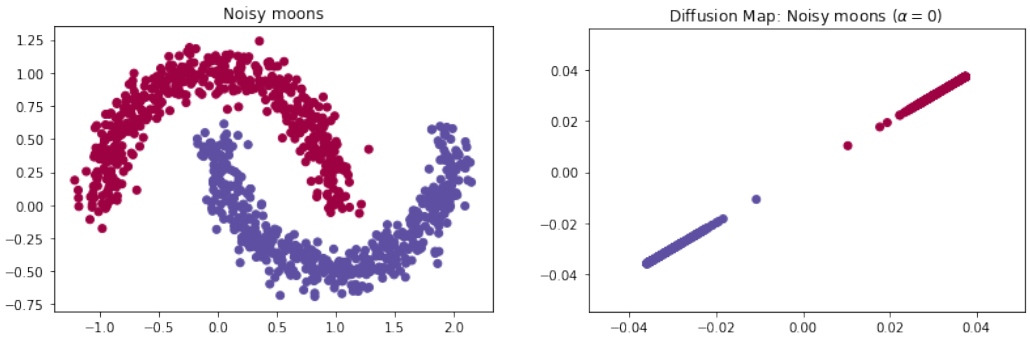

Figure 2.4.1: Spectral Clustering in the noisy moons dataset. Left: data in the original space. Right: The embedding computed via Diffusion Maps, with kernel scale $\sigma = 0.1234$ and $\alpha = 0$.

focused in the embedding computed via Laplacian Eigenmaps. However, note that Laplacian Eigenmaps is a particular case of Diffusion Maps, in which the anisotropic parameter α is set to 0 and the number of steps is $t = 0$. Given a dataset, the steps are the following:

1. Construct the similarity graph.
2. Compute the embedding, $Y \subset \mathbb{R}^d$, via Diffusion Maps or Laplacian Eigenmaps.
3. Cluster points $y_i \in Y$ with k-means using the Euclidean distance.

The main advantage of applying k-means in the embedding space instead of the original space is that, if the diffusion metric is right in the manifold, we will obtain better results. Close points in the manifold will also be close in the embedding space.

An illustration of the advantages of doing clustering with k-means in the embedding space is shown in Figure 2.4.1. While it is not possible to separate in two groups the image on the left with k-means (see Section 2.4.1), the Diffusion Map embedding provides a perfect separation for both targets. The noisy moons problem is linearly separable in the embedding space.

Chapter 3

Out-Of-Sample Extension

In the previous chapter we described some very foundational concepts covering the basic theory and tools that are needed in order to understand more advanced topics regarding manifold learning and dimensionality reduction when using Diffusion Maps. In this section, we will show different ways to extend the embedding given by Diffusion Maps to new patterns without re-doing the eigenanalysis over an extended similarity matrix, which is computationally expensive ($\mathcal{O}(N^3)$). From now on, the patterns to which we want to extend the embedding will be referred as OOS (out-of-sample). This chapter is divided into the next sections that will cover the following topics:

1. In Section 3.1 we introduce different ways to extend the embedding, depending on the target of the approximation. We will also introduce two ways to measure the low-rank approximation quality in order to compare the different approaches to OOS extensions.
2. In Section 3.2 we will explain how the Nyström method can be used to define the projections of OOS patterns.
3. In Section 3.3 we will introduce a recent development that aims to merge deep learning concepts with the extension of the embedding. In particular, a neural network will be trained to learn the encoding and decoding map.

3.1 Low-Rank Approximation

Predicting the embedding given by Diffusion Maps for an OOS example can be done following different options. The general set up, given a sample with N patterns, is as follows:

1. Select a subset of size N_L of the entire sample, i.e. a subset of landmarks patterns, and calculate the corresponding diffusion matrix, $P^{(L)}$, and its symmetric version, $A^{(L)}$. These patterns will compose the landmark subset $L = \{x_1, \ldots, x_{N_L}\}$.
2. Define an extension of the resulting encoding map to be applied to the remaining $N' = N - N_L$ patterns, which are seen as OOS examples.

Landmark selection is an important problem for which several solutions have been proposed [12, 13]. This work is focused only on the encoding and decoding extensions, so no analysis on how to choose the subset of landmarks will be carried out. Suppose we want to give the diffusion coordinates of a new point x. We could take two different approaches.

The first one is to learn an eigenvector extension for the OOS patterns, either from the diffusion matrix $P^{(L)}$, or from the symmetric operator $A^{(L)}$ defined in Section 2.3. Once we have the eigenvector extension, we can reconstruct the embedding for an OOS pattern $\hat{\Psi}_t(x)$, where t denotes, as usual, the number of steps. When extending the eigenvectors of $P^{(L)}$ to a point x, we will write $\hat{\psi}_i(x)$, $i = 1, \ldots, d$, where d denotes the embedding dimension. This is the approach followed in [14]. The extension of the Diffusion Map embedding, following the scheme of Definition 8, would be

$$\left(\hat{\Psi}_t(x)\right)_i = \lambda_i^t \hat{\psi}_i(x) \ ,$$

where λ_i is the i-th eigenvalue of $P^{(L)}$. It is also possible to learn the eigenvector extension at a point x for the symmetric matrix $A^{(L)}$. It may be useful to recall that $A^{(L)}$ and $P^{(L)}$ have the same eigenvalues (See Section 2.3, Equation (2.3.3)). In this case, the eigenvector extension is denoted by $\hat{\phi}_i(x)$, $i = 1, \ldots, d$. The extension to OOS patterns of the embedding would be

$$\left(\hat{\Psi}_t(x)\right)_i = \frac{\lambda_i^t}{\sqrt{d(x)}} \hat{\phi}_i(x) \ ,$$

where $d(x)$ is the degree of the OOS example x. The degree of an OOS pattern is computed by means of the kernel $k(x, y)$, from Equation (2.3.2), and the subset of landmarks as

$$d(x) = \sum_{x_k \in L} k(x_k, x) \ . \tag{3.1.1}$$

Another approach would be to learn directly a projection extension of the Diffusion Map, $\hat{\Psi}(x) \in \mathbb{R}^d$, which is the approach used in [1]. The eigenvector extension for the matrix $P^{(L)}$ can be recovered as

$$\hat{\psi}_i(x) = \left(\hat{\Psi}_t(x)\Lambda^{-t}\right)_i \ ,$$

where Λ is a diagonal matrix with the first d eigenvalues of $P^{(L)}$ or $A^{(L)}$ in the diagonal, which are the same for both matrices. In the case of $A^{(L)}$, the eigenvector extension would be

$$\hat{\phi}_i(x) = \sqrt{d(x)} \left(\hat{\Psi}_t(x)\Lambda^{-t}\right)_i \ ,$$

where $d(x)$ is calculated as in Equation (3.1.1). Identifying terms in the last two equations, we can relate the extensions of the eigenvectors of $P^{(L)}$ and $A^{(L)}$ as

$$\hat{\phi}_i = \sqrt{d(x)} \hat{\psi}_i \ . \tag{3.1.2}$$

When measuring the quality of the embedding to OOS patterns, there are several paths that can be followed. The first one is a straightforward approach, in which we calculate the Frobenius norm of the difference of the original matrices A and P, computed from the complete sample, and the one reconstructed using the landmark subset, so the best approach would be the one minimizing this quantity. This option is developed in Section 3.1.1. The second option, explained in Section 3.1.2, would be to directly measure the error between the original embedding and the predicted one. The problem with this method is that, when changing the data to compute de Diffusion Map, the embedding can change up to a rotation, which we have to compute.

3.1.1 Reconstructing the Kernel Matrix

This section provides an explanation on how to measure the low-rank approximation quality of the embedding by reconstructing the matrix A. Recall that the matrix A is built using a measure of similarity between points, $k(x_i, x_j)$, which is a symmetric and positive definite kernel, as

$$A_{ij} = a(x_i, x_j) = \frac{k(x_i, x_j)}{\sqrt{d(x_i)}\sqrt{d(x_j)}},$$

where $d(x_i)$ and $d(x_j)$ are the degrees of the patterns, as shown in Equation (2.3.2). In Section 2.3 it was shown that the matrix A has a discrete sequence of eigenvalues $\lambda_0, \ldots, \lambda_{N-1}$, and eigenvectors, $\psi_0, \ldots, \psi_{N-1}$, so we can write $A = U\Lambda U^T$, where $\Lambda_{ii} = \lambda_i$ is a diagonal matrix and U contains the eigenvectors of A in columns, which compose an orthonormal basis. Given a landmark subset L, A can be decomposed as follows

$$A = \begin{pmatrix} A^{(L)} & B^T \\ B & C \end{pmatrix} \tag{3.1.3}$$

where $B_{pq} = a(x_p, x_q)$ contains the kernel values computed for the landmark patterns and the testing subset; and $A^{(L)}$ is the matrix calculated for the landmark subset. That is to say, given a pair $x_i, x_j \in L$,

$$(A^{(L)})_{ij} = a(x_i, x_j) \ .$$

Since $A^{(L)}$ is also symmetric and positive definite, it has an eigenvalue decomposition. Following the notation, we will write $A^{(L)} = U^{(L)}\Lambda^{(L)}\left(U^{(L)}\right)^T$, obtaining

$$A = \begin{pmatrix} U^{(L)}\Lambda^{(L)}\left(U^{(L)}\right)^T & B^T \\ B & C \end{pmatrix} \ .$$

Let $\hat{U}$ be the matrix with a possible OOS extension of the eigenvectors for the test set, which is composed of OOS patterns. Then, we can write an approximation for A as

$$\begin{aligned} \hat{A} &= \begin{pmatrix} U^{(L)} \\ \hat{U} \end{pmatrix} \Lambda^{(L)} \begin{pmatrix} \left(U^{(L)}\right)^T & \hat{U}^T \end{pmatrix} \\ &= \begin{pmatrix} U^{(L)}\Lambda^L\left(U^{(L)}\right)^T & U^{(L)}\Lambda^{(L)}\hat{U}^T \\ \hat{U}\Lambda^{(L)}\left(U^{(L)}\right)^T & \hat{U}\Lambda^{(L)}\hat{U}^T \end{pmatrix} \ . \end{aligned}$$

The error will be measured with the Frobenius norm of the difference as

$$ERR = \left\| A - \hat{A} \right\|_F \ , \tag{3.1.4}$$

where

$$\|X\|_F = trace(X^T X),$$

for any matrix X. Depending on the method used to predict the embedding for an OOS pattern, this error is simplified. This fact will be developed in further sections.

3.1.2 Mean Squared Error of the Encoding

This section provides an explanation on how to measure the low-rank approximation quality of the embedding in terms of the mean squared error between the OOS extension and

the true embedding. That is the embedding computed via Diffusion Maps when using the complete sample, i.e. the OOS examples and the subset of landmarks. The process is explained in [11]. It is necessary to calculate a rotation matrix due to the possible transformations that suffers the embedding when training with different data points. Given the original embedding for the N examples, $\Psi \in \mathbb{R}^{N\times d}$, and the approximation $\hat{\Psi} \in \mathbb{R}^{N\times d}$ for the OOS examples, the procedure is as follows:

- Define $S_{ij} = \sum_k \Psi_i(x_k)\hat{\Psi}_j(x_k)$, with x_k in the landmark subset and $i, j = 1, \ldots, d$.
- Compute the singular value decomposition of S, $S = U\Lambda V^T$.
- Compute the rotation between Ψ and $\hat{\Psi}$ as the matrix $R = VU^T$.
- Define $SE_q = \left\| \Psi(x_q) - R\hat{\Psi}(x_q) \right\|^2$, where x_q is in the test set.
- Finally, the MSE for the test set would be the average of the SE_q, $q = N_L, \ldots, N$.

This work is focused on measuring the quality of the OOS extensions by means of the reconstruction error of the kernel matrix. This other approach may be explored in future work.

3.2 Nyström's Encoding

This section is divided in two parts. The first part, Section 3.2.1, will explain the Nyström's approach to approximate a matrix built for an arbitrary symmetric and positive definite kernel, $k(x, y)$. In the second part, we will adapt this method to give an OOS extension for Diffusion Maps.

3.2.1 Nyström's Low-Rank Approximation

The Nyström encoding [4, 7] is useful when trying to approximate the kernel matrix, K, from the one calculated from a landmark sample $L = \{x_1, \ldots, x_{N_L}\}$, by means of a symmetric and positive definite kernel

$$k(x, y) = \sum_{i\geq 1} \lambda_i \phi_i(x)\phi_i(y) \ ,$$

where $\lambda_i \geq 0$. The matrix $K^{(L)}$ associated to $k(x, y)$ for the given landmark sample L is

$$K^{(L)}_{ij} = k(x_i, x_j), \text{ with } x_i, x_j \in L.$$

Hence, $K^{(L)}$ is symmetric and positive definite and, by the spectral theorem, we can write

$$K^{(L)}V^{(L)} = V^{(L)}\Lambda^{(L)} \ ,$$

where $\Lambda^{(L)}$ is a diagonal matrix with diagonal $\lambda^{(L)}_1 \geq \lambda^{(L)}_2 \geq \ldots \geq \lambda^{(L)}_N \geq 0$ and $V^{(L)}$ is the matrix with the eigenvectors in columns. That is to say,

$$\sum_{k=1}^{N_L} k(x_j, x_k)V^{(L)}_{ki} = \lambda^{(L)}_i V^{(L)}_{ji} \ .$$

or, equivalently,

$$V_{ji}^{(L)} = \frac{1}{\lambda_i^{(L)}} \sum_{k=1}^{N_L} k(x_j, x_k) V_{ki}^{(L)} \ . \tag{3.2.1}$$

We can identify each row of $V^{(L)}$ with the image of each example in L. So, for any $x_j \in L$, we define the function

$$V_i(x_j) = V_{ji}^{(L)}, \ 1 \leq i \leq d \ .$$

That is to say, $V_i(x_j)$ denotes the i-th component of the j-th row of $V^{(L)}$. Equation (3.2.1) suggests to define the Nyström extension to a new $x \notin L$ through the kernel values $k(x, x_k)$, $x_k \in L$, as

$$\hat{V}_i(x) = \frac{1}{\lambda_i^{(L)}} \sum_{k=1}^{N_L} k(x, x_k) V_{ki}^{(L)} \ .$$

3.2.2 Nyström's Method and Diffusion Maps

As shown above, the Nyström's encoding can be applied as long as the kernel is symmetric and definite positive. Therefore, in the context of Diffusion Maps, it can be applied to the kernel $a(x, y)$ to extend the eigenvectors of A to OOS examples x as

$$\hat{\phi}_i(x) \simeq \frac{1}{\lambda_i} \sum_{k=1}^{N_L} a(x, x_k) \phi_i(x_k) \ ,$$

where $\phi_i(x_k)$ denotes the k-th element of the i-th eigenvector of A. Note that we write $\phi_i(\cdot)$ to refer both to an eigenfuction of the operator associated to the kernel $a(x, y)$ and to an eigenvector of A. This is because the theory of Diffusion Maps is developed in the continuous case, but when working with finite datasets, the eigenfunctions are approximated by eigenvectors.

In addition, it is easy to check that the Nyström's method can be also directly applied to P, obtaining the equation

$$\hat{\psi}_i(x) \simeq \frac{1}{\lambda_i} \sum_{k=1}^{N_L} p(x, x_k) \psi_i(x_k) \ .$$

To prove it, we have

$$\begin{aligned}
\hat{\psi}_i(x) &= \frac{\hat{\phi}_i(x)}{\sqrt{d(x)}} \\
&= \frac{1}{\lambda_i \sqrt{d(x)}} \sum_{k=1}^{N_L} a(x, x_k) \phi_i(x_k) \\
&= \frac{1}{\lambda_i} \sum_{k=1}^{N_L} \frac{k(x, x_k)}{d(x)\sqrt{d(x_k)}} \phi_i(x_k) \\
&= \frac{1}{\lambda_i} \sum_{k=1}^{N_L} p(x, x_k) \frac{\phi_i(x_k)}{\sqrt{d(x_k)}} \\
&= \frac{1}{\lambda_i} \sum_{k=1}^{N_L} p(x, x_k) \psi_i(x_k) \ ,
\end{aligned}$$

where we have used the relation between the extensions of the eigenvectors of Equation (3.1.2), $\hat{\phi}_i = \sqrt{d(x)}\hat{\psi}_i$.

Equivalently, the OOS extension for the matrix containing the eigenvectors of A can be written as

$$\hat{U}_{qi} = \hat{\phi}_i(x_q) = (BU^{(L)}(\Lambda^{(L)})^{-1})_{qi} \ ,$$

where $\hat{U}$ denotes the approximation for U, $q = N_L, \ldots, N$ are the indices of the OOS set and B, $U^{(L)}$ and $\Lambda^{(L)}$ are the same as in Equation (3.1.3). The low-rank approximation $\hat{A}$ is then

$$\begin{aligned}
\hat{A} &= \begin{pmatrix} U^{(L)} \\ \hat{U} \end{pmatrix} \Lambda^{(L)} \begin{pmatrix} \left(U^{(L)}\right)^T & \hat{U}^T \end{pmatrix} \\
&= \begin{pmatrix} A^{(L)} & U^{(L)}\Lambda^{(L)}(\Lambda^{(L)})^{-1}(U^{(L)})^T B^T \\ BU^{(L)}(\Lambda^{(L)})^{-1}\Lambda^{(L)}\left(U^{(L)}\right)^T & BU^{(L)}(\Lambda^{(L)})^{-1}\Lambda^{(L)}(\Lambda^{(L)})^{-1}(U^{(L)})^T B^T \end{pmatrix} \\
&= \begin{pmatrix} A^{(L)} & B^T \\ B & B(A^{(L)})^{-1}B^T \end{pmatrix} \ .
\end{aligned}$$

Thus the reconstruction error when using the Nyström approximation [12] to extend the eigenvector of A would be, substituting in Equation (3.1.4),

$$ERR_{nys} = \left\| C - B(A^{(L)})^{-1}B^T \right\|_F^2 \ , \tag{3.2.2}$$

where

$$B_{kq} = a(x_k, x_q) = \frac{k(x_k, x_q)}{\sqrt{d(x_k)}\sqrt{d(x_q)}} \ , \text{ with } x_k \in L, x_q \notin L \text{ , and}$$

$$C_{pr} = a(x_p, x_r) = \frac{k(x_p, x_r)}{\sqrt{d(x_p)}\sqrt{d(x_r)}} \ , \text{ with } x_p, x_r \notin L \ .$$

3.3 Diffusion Nets

The aim of Diffusion Nets [1] is to employ deep learning from a manifold learning perspective. The authors address three problems: OOS embeddings of new points, a pre-image solution that can include regularization, and outlier detection on test data. The proposal is to train two neural networks, which will be referred as encoder and decoder, and combine them in an autoencoder. Of these three we will study the encoder solution as an OOS extension method.

3.3.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are networks composed of connected units, which are called neurons. A particular example is the Multilayer Perceptron (MLP), a network with fully connected layers. An example of a MLP can be seen in Figure 3.3.1. The output of each layer is computed as

$$o^{(l+1)} = h(W^{(l)}o^{(l)} + b^{(l)}) \ ,$$

where $h(\cdot)$ is a non-linear function called activation, $W^{(l)}$ are the weights that connect the neurons in the layer l with those in the layer $l+1$, $o^{(l)}$ is the output of the previous layer

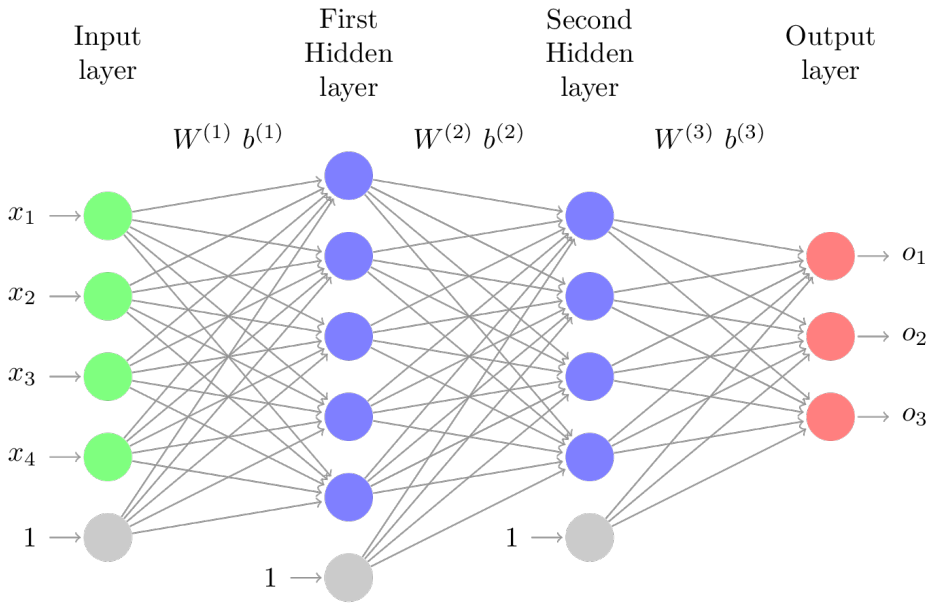

Figure 3.3.1: Standard Multilayer Perceptron with two hidden layers.

and $b^{(l)}$ is a bias term. The set of weights is defined as $\Theta = \{W^{(l)}, b^{(l)}\}$. We denote by $\mathcal{L}$ the number of layers in the network and by s_l the number of neurons in the layer l, $1 \leq l \leq \mathcal{L}$. The experiments carried out in [1] are done using the sigmoidal as the activation for the hidden layers. This is

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

Other choices may include $h(z) = \tanh(z)$ or the rectified linear units,

$$h(z) = \text{ReLU}(z) = \max\{0, z\} \ .$$

Particularly, these kind of networks can be used for regression. Following the standard for regression, the activation functions for the output layer are the identity. The training set will be composed of N observations

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\} \ ,$$

where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}^m$, $i = 1, \ldots, N$. The MLP is, thus, a function

$$\begin{aligned} o : \mathbb{R}^m &\to \mathbb{R}^d \\ x &\mapsto o(x, \Theta) \ , \end{aligned}$$

where m denotes the input dimension, d is the output dimension and Θ is the set of weights of the neural network. When we refer to the complete output matrix, which contains the image by the MLP of all the patterns in our dataset, we will write O. These parameters are chosen to minimize a loss function. For this purpose, gradient descent based methods are used. The gradient is computed with the back propagation algorithm. For regression problems the standard regularized loss function is

$$J^{REG}(\Theta) = \frac{1}{2N} \sum_{i=1}^{N} \|o(x_i, \Theta) - y_i\|^2 + \frac{\mu}{2} \sum_{l=1}^{\mathcal{L}-1} \left\|W^{(l)}\right\|^2 \ , \tag{3.3.1}$$

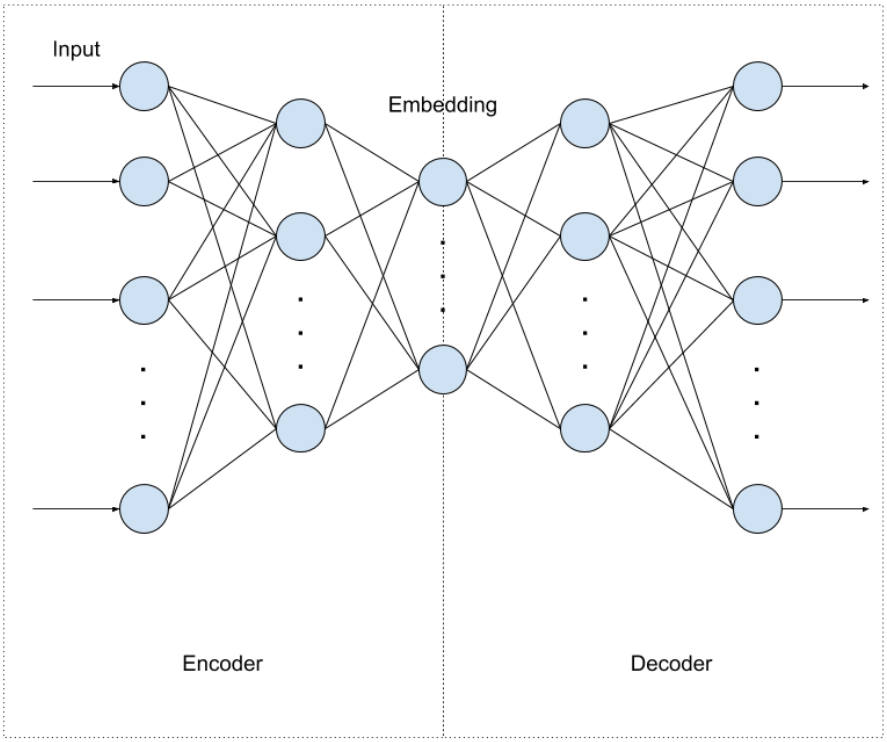

Figure 3.3.2: Autoencoder. Left: encoder with one hidden layer. Right: decoder with one hidden layer.

where μ is a parameter to control the importance of the regularization term. This term aims to avoid overfitting.

3.3.2 OOS Example Extension: the Encoder

When using a neural network to predict the embedding for OOS examples, we will assume the data lies on a smooth, compact, d-dimensional Riemannian manifold. The Diffusion Map for the training set embeds the landmark subset $L = \{x_1, \ldots, x_{N_L}\}$ into the Euclidean space $\mathbb{R}^d$. The diffusion embedding is denoted by $\Psi \in \mathbb{R}^{N \times d}$, which is a matrix whose rows correspond to the embeddings computed for the training points.

Given new test points, $\mathcal{X}' = \{x_q\}_{q=N_L,\ldots,N}$, the aim is to calculate an approximation as close as possible to the true embedding. For each test point $x_q \in \mathcal{X}'$, we will write $\hat{\Psi}(x_q)$ to denote the embedding approximation. It is also desirable that the approximation preserves the original properties of the embedding. With this purpose, the encoder is designed as an MLP, minimizing the L_2 loss between the diffusion embedding Ψ and the output of the net, which is denoted by $O^e \in \mathbb{R}^{N \times d}$. Since the coordinates of the Diffusion Map are eigenvectors of the random walk matrix on the data, P, the j-th column of O^e should fulfill

$$PO^e_j \simeq \lambda_j O^e_j \, , 1 \leq j \leq d \, . \tag{3.3.2}$$

where, as usual, λ_j denotes the j-th eigenvalue of P.

The architecture of the encoder is shown in the left part of Figure 3.3.2 and the algorithm to train the encoder is shown in Algorithm 1. The set up is: define a MLP with $\mathcal{L}$

hidden layers whose output layer is a regression layer (i.e. the identity is its activation) and minimize the loss function defined in (3.3.1). However, the authors of [1] proposed a modified loss function,

$$J^e(\Theta) = J^{REG}(\Theta) + J^{EV}(\Theta) \ , \tag{3.3.3}$$

where J^{REG} is the standard regularized loss for regression of Equation (3.3.1) and

$$J^{EV}(\Theta) = \frac{\eta}{2N_L} \sum_{j=1}^{d} \left\| (P - \lambda_j \mathcal{I}_{N_L \times N_L}) O_j^e \right\|^2 \ , \tag{3.3.4}$$

where η is an optimization cost parameter and $\mathcal{I}_{N_L \times N_L}$ denotes the identity matrix of dimension N_L. They also provide a calculation for the gradient of this loss function with respect to the output layer, which is

$$\nabla J^{EV} = \frac{\eta}{N_L} \sum_{j=1}^{d} (O_j^e)^T (P^T - \lambda_j \mathcal{I}_{N_L \times N_L})(P - \lambda_j \mathcal{I}_{N_L \times N_L}) \ .$$

The aim of the penalization of Equation (3.3.4) is to force the output of the neural network to be eigenvectors of the Diffusion Matrix P, so the embedding still conserves its initial properties. The value of η has to be chosen carefully because, otherwise, the output will be forced to be 0. In contrast to general constraints, this restriction needs output values of the different test points together. For this reason, using this term can lead to computational problems, as the new gradient cannot be decomposed to use online training because the loss function needs the values predicted to all the examples in order to be evaluated. In [1] it is shown that incorporating this term does not lead to extremely better solutions, so in this work we will not consider (3.3.4).

Algorithm 1: Encoder

Input: Training set X, diffusion embedding Ψ for X, number of layers $\mathcal{L}$, neurons per layer s_l, $l = 1, \ldots, \mathcal{L}$.

Output: *None*

1: Initialize the weights Θ of the encoder and set X as input and Ψ as target.
2: Train the network minimizing (3.3.3) with back-propagation

Once we have a prediction for the test set, there are different ways to measure the quality of the approximation. In the original work [1], they used the approach explained in Section 3.1.2, measuring the MSE made by the encoder using LOOCV (Leave One Out Cross Validation). However, this approach does not verify that the predicted embedding fulfills the property of Equation (3.3.2) and we may lose the eigenvalue structure of the embedding.

The authors not only proposed in [1] an extension of the embedding to OOS using neural networks. They also provide a theoretical bound on the error rate for approximating eigenfunctions of the Laplacian using a MLP with sigmoids activations. Suppose $\mathcal{M} \subset \mathbb{R}^n$ a d-dimensional manifold.

Definition 9 (Locally bi-Lipschitz metric)**.** *We say $\rho : \mathcal{M} \times \mathcal{M} \to \mathbb{R}^+$ is a locally bi-Lipschitz metric with respect to the Euclidean metric if $\forall \epsilon \ \exists \delta$ such that $\|x - y\| \leq \delta$ then,*

$$(1 - \epsilon) \|x - y\| \leq \rho(x, y) \leq (1 + \epsilon) \|x - y\| \ .$$

Theorem 6. *Let $\mathcal{M}$ be a smooth compact Riemmanian submanifold of $\mathbb{R}^m$ equipped with a locally bi-Lipschitz metric.*

- *Let B_r be a ball of radius r such that $\mathcal{M} \subset B_r$. That is to say, $\mathcal{M}$ is bounded.*
- *Let ψ be an eigenfunction of the Laplacian of $\mathcal{M}$, with eigenvalue λ.*
- *Let f be an extension of ψ defined as*

$$f(x) = \exp\left(-\lambda \left\| x - P^2_{\mathcal{M}(x)} \right\|\right) \psi\left(P_M(x)\right) \ ,$$

where $P_M(x) = argmin_{y \in \mathcal{M}} \|x - y\|^2$.

Then, there exists a linear combination of K sigmoid units, $f_k(x)$, such that

$$\|f(x) - f_k(x)\|_2 \leq \frac{C}{\sqrt{K}} \ .$$

Corollary 1. *Under the same conditions of Theorem 6, let $\psi_1, \ldots, \psi_d$ be the eigenfunctions of the diffusion operator and $f(x) = (f_1(x), \ldots, f_d(x))$ their respective extensions. Then, there exists a neural network with one hidden layer, $K \times d$ sigmoid units and output $o(x) \in \mathbb{R}^d$ such that*

$$\|f(x) - o(x)\|_2 \leq \frac{C}{\sqrt{K}} \ .$$

Note that there may be better bounds. The result, whose full derivation can be seen in [1], is theoretical and it depends on the architecture and the activations of the net. Hence, the bound might not be reached, as the loss function is not convex and back propagation does not guarantee convergence to the global minimum.

3.3.3 Decoder

The network that tries to obtain the initial point, $x \in \mathbb{R}^m$, given its embedding o, is called a decoder. The architecture is shown in the right part of Figure 3.3.2. It is also a standard neural network for regression trained with the loss shown in Equation (3.3.1). The eigenvector penalization does not make sense in this architecture.

The set up is explained in Algorithm 2. It is important to note that the recovered points x of the decoder may not exist in the available dataset, so this network can be used to increase the amount of data, a technique known as Data Augmentation. Another application would be, related to clustering, to pullback the centroids calculated by k-means in the embedding space (see Section 2.4).

Due to time restrictions, the decoder has not been implemented in this work.

3.3.4 Autoencoder

Having trained the encoder and the decoder, both networks can be stacked together to obtain an autoencoder. The architecture is shown in Figure 3.3.2. One application is

Algorithm 2: Decoder

Input: Training set X, diffusion embedding Ψ for X, number of layers $\mathcal{L}$, neurons per layer s_l, $l = 1, \ldots, \mathcal{L}$.

Output: *None*

1: Initialize the weights Θ of the encoder and set Ψ as input and X as target.
2: Train the network minimizing (3.3.1) with back-propagation

Algorithm 3: Autoencoder

Input: Trained encoder and decoder

Output: *None*

1: Stack decoder trained as in Algorithm 2 on top of encoder trained as in Algorithm 1.
2: Calculate the average reconstruction error ϵ.

anomaly detection. Denoting the output of the autoencoder by $r(\cdot)$, the training data is used to calculate the average reconstruction error as

$$\epsilon = \frac{1}{N_L} \sum_{i=1}^{N_L} \|x_i - r(x_i)\|^2 \ ,$$

where $x_i \in L$. For a new given point x, we will say it is an outlier if

$$\|x - r(x)\|^2 \geq \rho\epsilon \ ,$$

where ρ is a constant determined by the user.

Another application of the autoencoder is noise reduction. Noise in data is related to eigenvalues with small value, which are not considered when doing the embedding. So noisy points related to the same original point in the manifold should have an identical embedding. This original point is recovered by the decoder, while suppressing noise.

Chapter 4

Experiments

After covering the concepts and ideas of Spectral Embedding and Diffusion Maps and some of the possibilities to extend the embedding to out-of-sample (OOS) examples, we will divide this chapter in the following sections:

1. In Section 4.1 we will review the available software and libraries to apply Diffusion Maps. We also propose three datasets, which will be used to test the OOS extensions.
2. In Section 4.2, we will test our implementation of Diffusion Maps.
3. In Section 4.3 the Nyström's encoding will be used to extend the embedding for OOS examples.
4. Lastly, in Section 4.4 we will extend the embedding using Diffusion Nets, measuring the error by reconstructing the Gramm Matrix.

4.1 Software and Datasets

The experiments carried out in this work have been done using standard libraries and well proven packages that are available in the web. The code has been implemented using Python as the programming language. Manifold Learning is a well known topic and the following packages contain implementations of different algorithms:

- megaman: Manifold Learning for Millions of Points [15] is a scalable manifold learning package implemented in Python. It has a front-end API designed to be familiar to scikit-learn but harnesses the C++ Fast Library for Approximate Nearest Neighbors (FLANN) and the Sparse Symmetric Positive Definite (SSPD) solver Locally Optimal Block Precodition Gradient (LOBPCG) method to scale manifold learning algorithms to large datasets.
- NumPy [16] is a Python package that contains an implementation of a powerful N-dimensional array object. It adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Name	Data Type	N	m	L
Helix	Continuous	2000	3	-
Red Wine	Continuous	1599	11	10
Vowel	Continuous	990	13	11

Table 4.1.1: Description of the datasets used in the experiments. N is the number of examples, m denotes the initial dimension and L is the number of different values that can take the target.

- scikit-learn [17] contains simple and efficient tools for data mining and data analysis and it is accessible to everybody, and reusable in various contexts. It contains efficient ways to compute the eigendecomposition of kernel matrices.

Even though megaman is a powerful library, it has not been tested with the last version of Python. Due to incompatibilities with the software installed in the computer in which the experiments have been carried out, we have discarded its use.

scikit-learn is the natural election to implement an efficient library to compute Diffusion Maps. It contains scalable ways to compute the kernel matrix of a given dataset. There are also methods to calculate the eigendecomposition of symmetric kernel matrices accurately and efficiently. For these reasons, we have implemented the class `DiffusionMap`, which was not available in any of the used packages. The construction of the kernel matrix and its eigenanalysis harnesses the power of scikit-learn and NumPy. Matrix operations are always carried out using NumPy.

Regarding to the experiments related to the encoder and the Diffusion Nets, which are MLP (Multilayer Percentron) regressors, we will use the implementation of the class `MLPRegressor` of scikit-learn, which contains an efficient implementation that optimizes the standard regularized squared-loss using LBFGS or stochastic gradient descent. It also incorporates a variety of activation functions and it is compatible with `GridSearchCV`, class that makes an exhaustive search over specified parameter values for an estimator.

Regarding the datasets, three problems have been chosen. The first one, the helix dataset, is a synthetic experiment. The other two datasets are real life examples. From them we will obtain three embeddings that we will try to extend to OOS examples in order to provide an estimation of the quality of the extension given by Nyström or Diffusion Nets. They are summarized in Table 4.1.1, which contains information about the data type, the number of examples available, the dimension and the number of classes of each dataset. Next, we provide descriptive information about them.

- **Helix.** This is a synthetic experiment proposed in [1]. It is a 1-dimensional manifold that lives in the Euclidean space $\mathbb{R}^3$. It has been previously introduced in the Section 2.3.1 of this work. The parametrization of the curve is:

$$\begin{aligned} x_i &= cos(\theta_i) + \epsilon \\ y_i &= sen(2\theta_i) + \epsilon \\ z_i &= sen(3\theta_i) + \epsilon \ , \end{aligned} \tag{4.1.1}$$

  where ϵ denotes Gaussian white noise with standard deviation $\sigma_r = 0.5$. The number of generated samples is $N = 2000$.

- **Red Wine.** This dataset is related to red variants of the Portuguese *Vinho Verde* wine. For more details, see [18]. It contains $N = 1599$ instances, each one of $m = 11$ features and an output attribute that denotes the quality of the wine.
- **Vowel.** This dataset contains speaker independent recognition of the 11 steady state vowels of British English. It contains $N = 990$ instances, each one of $m = 10$ features. For more details, see [19].

4.2 Computing Diffusion Maps

This section contains some analysis on the embeddings computed via Diffusion Maps of the datasets described in the previous section (see Table 4.1.1). Recall that the Similarity Graph used in Diffusion Maps is always a fully connected graph, so there will be just one connected component. The notion of similarity is given by a positive definite kernel, $k(x, y)$. In our experiments, we will use the Gaussian kernel

$$k(x, y) = \exp\left(\frac{-1}{2\sigma}\|x - y\|^2\right) ,$$

where σ is the scale of the kernel. In Section 2.3 we introduced the Anisotropic Diffusion and it was proved that setting $\alpha = 1$ would recover the geometry of the manifold, independently of the density of the sample. Because of that, the experiments are carried out with that configuration. After normalization through the degrees, we obtain the anisotropic kernel $k^{(\alpha)}(x, y)$, which will help us compute the diffusion matrix P.

Diffusion Maps is based in the eigenalysis of the Diffusion Matrix P, which is related with the probability of transition in a random walk defined over the set of points. However, P is not symmetric and, in order to apply theory of Section 3, we will always work with A, the symmetric matrix, whose eigenvectors are related to those of P by means of the degree matrix D. Given (λ, ϕ) an eigenvalue and an eigenvector of A, then $(\lambda, \psi = D^{-1/2}\phi)$ are an eigenvalue and a right eigenvector of P. Finally, given a point x we can compute its embedding as

$$\Psi_t(x) = \begin{pmatrix} \lambda_1^t \psi_1(x) \\ \lambda_2^t \psi_2(x) \\ \vdots \\ \lambda_d^t \psi_d(x) \end{pmatrix} ,$$

where t is the number of steps considered in the random walk over the data. For each datasets we will analyze the spectrum decay of A, so the most appropriate values for σ, t and d are chosen, given the criterium of Equation (2.3.1)

$$d = s(\delta, t) = \max\{l \in \mathbb{N} : |\lambda_l|^t > \delta|\lambda_1|^t\} ,$$

where δ controls the precision of the approximation of the Diffusion Distance D_t in the embedding space. We will set $\delta = 0.1$ in every experiment.

4.2.1 Helix dataset

The Helix dataset was proposed in [1], so our first approach is to duplicate the experiment carried out by the authors. Figure 4.2.1 contains the helix generated by the Equation (4.1.1)

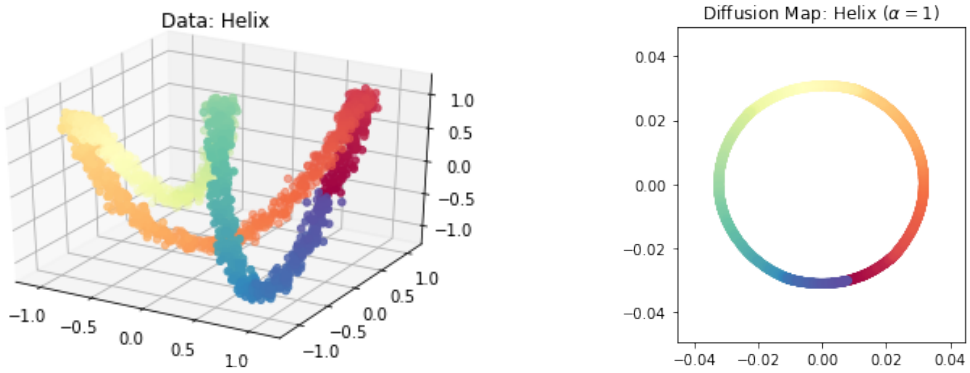

Figure 4.2.1: Replication of Helix dataset embedding. Left: original curve. Right: the Laplace–Beltrami approximation ($\alpha = 1$) following the set up given in [1].

and the embedding computed via Diffusion Maps with $\sigma = 0.1$ for the scale parameter and $t = 1$ for the number of steps, which is the authors' proposal. The 2-dimensional embedding is a smooth circle. The Helix will be always represented graphically with a color code that is related to the value of the parameter θ at each point. As an idea, we will consider a good embedding the one that keeps points with similar colors near (in an Euclidean sense).

Even though the curve is perfectly unrolled with the proposed configuration, the authors do not provide an explanation for the selection of σ and t.

A proposal to select σ is to analyze the matrix of distances between each pair of points in the dataset. The parameter for the scale of the kernel can be set by taking into account certain percentile of the distances, which will be denoted by p. Note that the percentile chosen might depend on the density of the sample and the dimension. It is well known that the volume of a ball of radius r decreases roughly when increasing the dimension, which is typically the case to apply dimensionality reduction. Higher dimensions may require to consider wider neighborhoods. In particular, if we set $r = 1$, we can compute the volume of the d-sphere as:

$$V_d = \frac{\pi^{d/2}}{\Gamma(d/2+1)} \,,$$

where d denotes the dimension of the sphere and $\Gamma(\cdot)$ is the Gamma function. Note that Γ can be thought as an extension of the factorial to noninteger arguments, so it dominates the term $\pi^{d/2}$ when $d \to \infty$. That is to say, depending on the dimension we may need to increase the scale of the kernel, so the same number of neighbors are taken into account by the kernel function. Otherwise, the kernel matrix will be close to a diagonal matrix.

The spectral decay when varying the value of p with $t = 1$ fixed can be seen in Figure 4.2.2. Recall that the multiplicity of the eigenvalue 1 is related to the number of connected components in the graph. As the width of the kernel increases, the weights in the graph also increase, and the graph becomes denser (weights that where practically zero before, now are larger). Because of that, setting small values for p or σ implies a flat spectrum. In the limit, if $\sigma \to 0$, the multiplicity of the eigenvalue 1 would be N, as every point would be a connected component. We can also see the resulting embeddings fixing $t = 1$ and varying the percentile of the distances in Figure 4.2.3. The Helix is perfectly unrolled choosing $p = 2$.

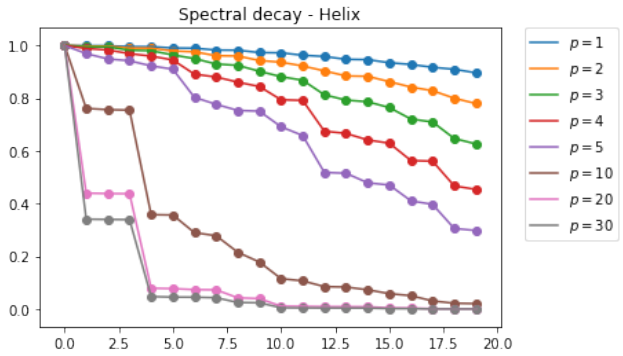

Figure 4.2.2: Spectral decay of the Helix for different values of σ and $t = 1$. In this case, σ is calculated as the p-th percentile of the distances.

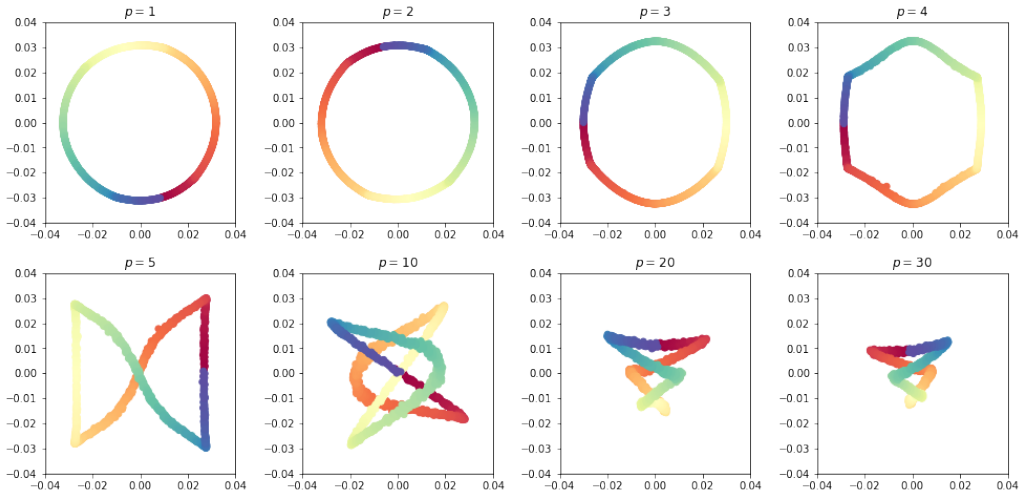

Figure 4.2.3: Embeddings for the Helix dataset for fixed $t = 1$ and different values of σ. In this case, σ is calculated as the p-th percentile of the distances.

In this experiment the embedding dimension has been set to two without any prior analysis, in spite of the fact that the actual dimension of the underlying manifold is actually one, as it can be parametrized with just one parameter (i.e. θ). It can be argued that, since we are working with a closed manifold, we need two dimensions to embed the data. Otherwise, the helix would be broken and points that are near in the original space would be mapped far away. In Section 2.3 we gave a criterion to select the dimension of the embedding depending on the spectral decay of the eigenvalues of the matrix $P = D^{-1}W$, where D is the degree matrix and W denotes the adjacency matrix. Recall that, given a precision δ, the embedding dimension would be

$$d = s(\delta, t) = \max\{l \in \mathbb{N} : |\lambda_l|^t > \delta |\lambda_1|^t\} .$$

Figure 4.2.4 shows the spectral decay of the eigenvalues given the kernel parameter $\sigma = 0.1$ (left) and the percentile $p = 2$ (right) for $t = 1$ fixed. The spot line is set to $0.1|\lambda_1|$, where λ_1 is the biggest eigenvalue different to 1. Obviously, if we follow the proposed criterium, the embedding dimension would be set to a value bigger than 20, which does not make sense, as the initial dataset is 3-dimensional. This is because the Helix dataset is synthetic, and, due to the high density of points it is very difficult to obtain a fast decay of the spectrum. There are other popular criteria in the literature. For example, we could set d as the number of

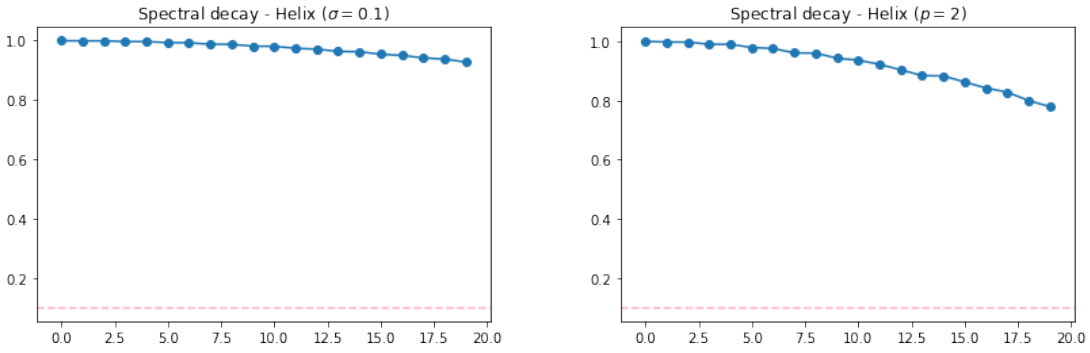

Figure 4.2.4: Spectral decay of the Helix dataset for $t = 1$. Left: the kernel scale is set to $\sigma = 0.1$. Right: the kernel scale is computed as the 2-th percentile of the distances. In both cases, the spot line indicates the value $0.1|\lambda_1|$.

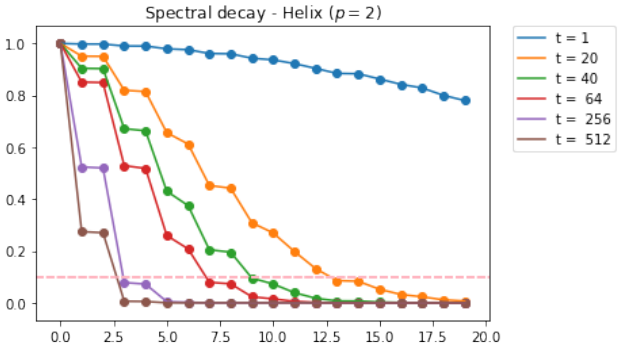

Figure 4.2.5: Spectral decay of the Helix for different values of t. In this case, σ is calculated as the 2-th percentile of the distances.

components in which there is not a significant gap in the spectrum [5]. However, we still cannot set $d = 2$, because there is not such a gap.

We can see that aiming to obtain a spectral decay big enough to select only two dimensions would result in a catastrophic embedding. However, we have just analyzed the spectral decay for different values of p but keeping $t = 1$ fixed. In Figure 4.2.5 we can see the spectral decay for different values of t, but keeping the scale parameter computed as the 2-th percentile of the distances. Note that the spectral decay is very slow for small values of t, so we might need to increase the kernel scale or set a higher value for t if we are aiming to obtain a 2-dimensional embedding. Recall that t is related with the probability of transition, as we are measuring the probability of traveling from a given point to another in t steps. As the helix dataset is very dense, it makes sense to consider a large number of steps.

As the number of samples is $N = 2000$, taking this percentile means that, normally, we consider that about 40 points are connected to a given one. That is, on average, for each point x we can find about 40 points y for which the value $k(x, y)$ is significantly different to zero. Following this criterion, for our final embedding, we will set $t = 40$, as t is related to the number of steps in the Markov chain of the random walk defined over the data. In any case, taking 40 steps in the Markov chain does not mean that we are traveling to the nearest 40 points. The relation between t and the percentile p is still not clear, and it may

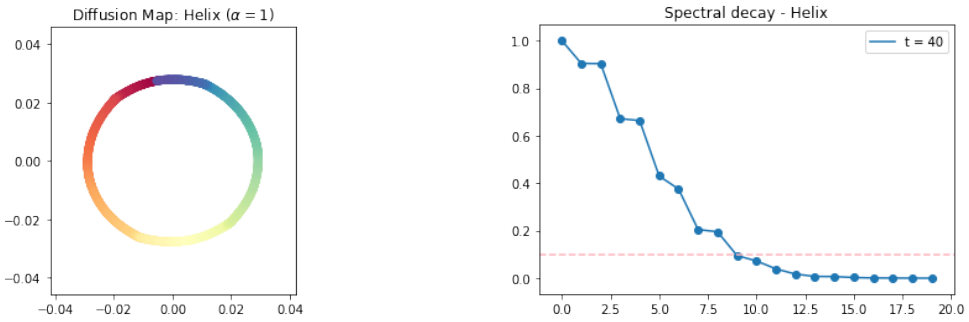

Figure 4.2.6: Embedding and spectral decay for the Helix dataset. The kernel parameter σ is chosen as the 2-th percentile of the distances, which is $\sigma = 0.1756$. The number of steps is set as $t = 40$.

be analyzed in further work.

Summing up, the embedding we will try to replicate for the Helix dataset will be the one given via Diffusion Maps, setting σ as the 2-th percentile of the distances and the number of steps $t = 40$. The final embedding and the spectral decay for the chosen parameters can be seen in Figure 4.2.6. Even though following the proposed criterion to select the dimension of the embedding does not work in this example, the are other criteria that are also popular in the literature. As we have said before, another possibility is to set d as the number of components in which there is not a significant gap in the spectral [5]. It is easy to see that, apart from the first eigenvalue (which is obviously 1), the next two are very similar, but a significant decay occurs when considering the fourth eigenvalue.

In the following experiments, we will make a similar analysis. First, we will study the spectrum and the embeddings for different values of σ in terms of the p-th percentile. Second, once we have fixed the kernel scale, we will fit t. The dimension of the embedding will be chosen following one of the two previously explained heuristics.

4.2.2 Red Wine dataset

The Wine dataset contains information about the quality of red and white Portuguese wine. In this work we will focus on the red wine variant. The dataset is composed of $N = 1599$ examples, each one with $m = 11$ features, and a label, an integer between 1 and 10 that indicates the wine quality. However, Table 4.2.1 shows that the problem is unbalanced. Furthermore, some values of the target never appear. For this reason, we have cleaned the dataset, and only the three main classes are kept for the analysis. They comprise the 94.93% of the total.

In addition, this dataset contains outliers. Figure 4.2.7 shows the boxplots of the dataset before and after removing extreme values. Outliers may affect the embedding, as an outlier can be seen as a connected component in the graph, since weights assigned to edges that connect any point with an outlier will be near zero; thus it is convenient to remove them. The proposal to remove these points is the following: for each attribute, compute the first and the third quantile (i.e. the percentiles 25 and 75 respectively). Let $Q1$ and $Q3$ be the vectors containing the first and the third quantiles for every attribute. Then, the interquartile range is defined as $IRC = Q3 - Q1$. We will say a given point x

Wine quality	Number of instances	Relative amount
1	0	0.0
2	0	0.0
3	10	0.00625391
4	43	0.03314572
5	681	0.42589118
6	638	0.39899937
7	199	0.12445278
8	18	0.01125704
9	0	0.0
10	0	0.0

Table 4.2.1: Distribution of the targets in the Red Wine dataset.

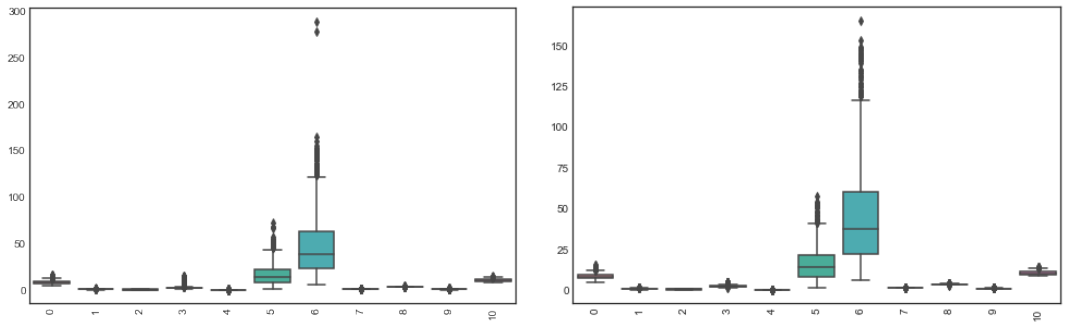

Figure 4.2.7: Boxplots of the Red Wine dataset. Left: original data. Right: data without outliers.

is an outlier if it verifies either one of the following conditions:

$$x > Q3 + 3 \times IRC, \text{ or,}$$
$$x < Q1 - 3 \times IRC \ .$$

After some research, we also realize that the dataset contains repeated examples. As the algorithms addressed in this work are related to kernel matrices and spectral decompositions we need to remove these entries, so non-invertible kernel matrices are avoided. After applying this, we are in conditions to analyze the decay of the spectrum for the Red Wine dataset, which, after the cleaning process, contains $N = 1175$ examples.

The spectral decay for different values of p, the percentile used to calculate the scale of the kernel, and $t = 1$ is shown in Figure 4.2.8. For small values of p the spectrum decays gradually, as the number of connected components increases. In the limit, setting $p \to 0$, the graph would contain N connected components. This phenomenon has already been observed in the previous section when studying the spectral decay of the Helix dataset.

The 3-dimensional embeddings computed via Diffusion Maps for the Red Wine and different values of p and $t = 1$ can be seen in Figure 4.2.9. For $p = 15$ the dataset is embedded into a smooth $3D$ curve. In addition, the spectral decay presents a significant gap when $p = 15$ between the third and the fourth eigenvalue (excluding the first one, which is always 1). This may indicate that $d = 3$ features could be enough to represent the data in the embedding dimension.

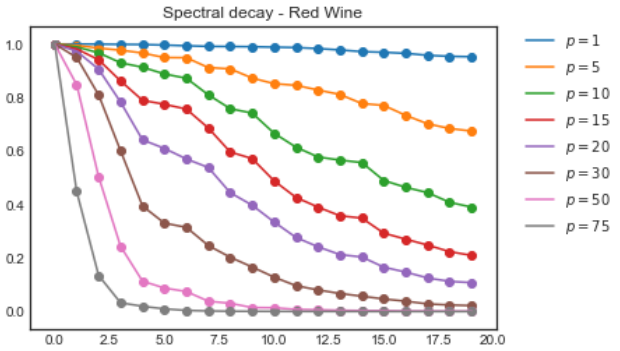

Figure 4.2.8: Spectral decay of the Red Wine dataset for different values of σ and $t = 1$. In this case, σ is calculated as the p-th percentile of the distances.

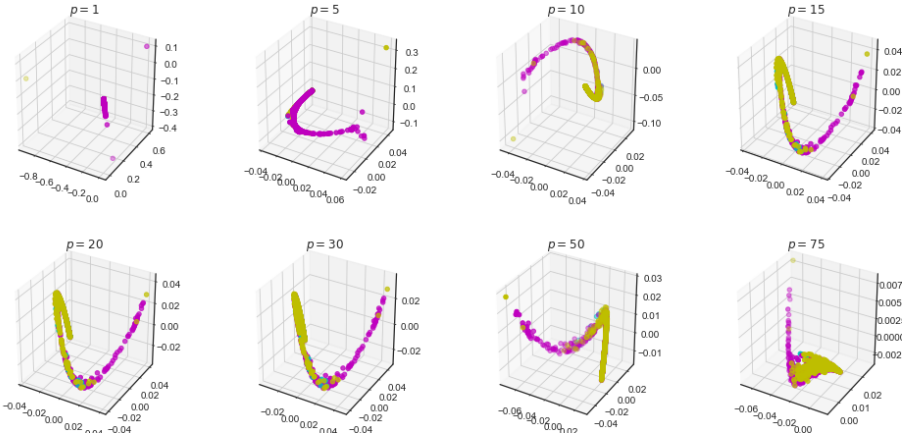

Figure 4.2.9: Embedding computed for the Red Wine dataset for fixed $t = 1$. In this case, σ is calculated as the p-th percentile of the distances.

After setting $p = 15$, we might study how the spectrum changes with t. Figure 4.2.10 shows the spectral decay for a fixed $p = 15$ and different values of t. The spot line is $0.1|\lambda_1|$, where λ_1 is the biggest eigenvalue of P different to 1. The plot indicates that setting $t = 16$ steps is enough to select 3 features in the embedding space. Recall that after the cleaning process, the Red Wine dataset contains $N = 1175$ examples. If we were in dimension 1, taking the 15-th percentile of the distances means that, normally, we are considering that almost 175 points are similar to a given one, number far away from 16, which is the number of steps we are taking into account. Actually, the Red Wine dataset has dimension 11, so, due to the curse of dimensionality, the number of neighbors decreases when keeping the scale of the kernel constant. For this reason, we are forced to select a smaller value for t, which is the number of steps in the random walk defined over the data. This means that we will consider that two points are nearby in the initial space (the manifold) if we can travel between them in less than 16 steps.

Finally, Figure 4.2.11 shows the final embedding computed via Diffusion Maps for the Red Wine dataset. Recall that even though the original problem was composed of 10 different classes, it has been reduced to 3 types of wine quality. The configuration parameters are set to $p = 15$ and $t = 16$. In the following sections we will try to extend this embedding with OOS extension methods.

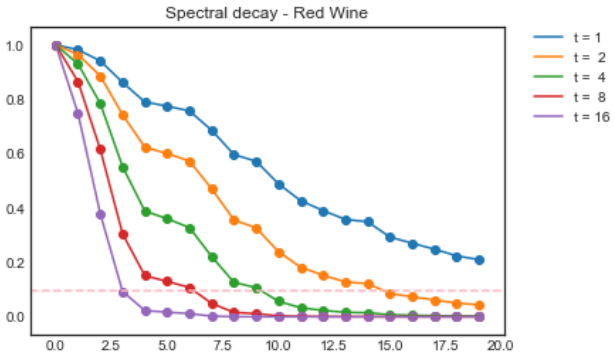

Figure 4.2.10: Spectral decay of the Red Wine dataset for different values of t. In this case, σ is calculated as the 15-th percentile of the distances.

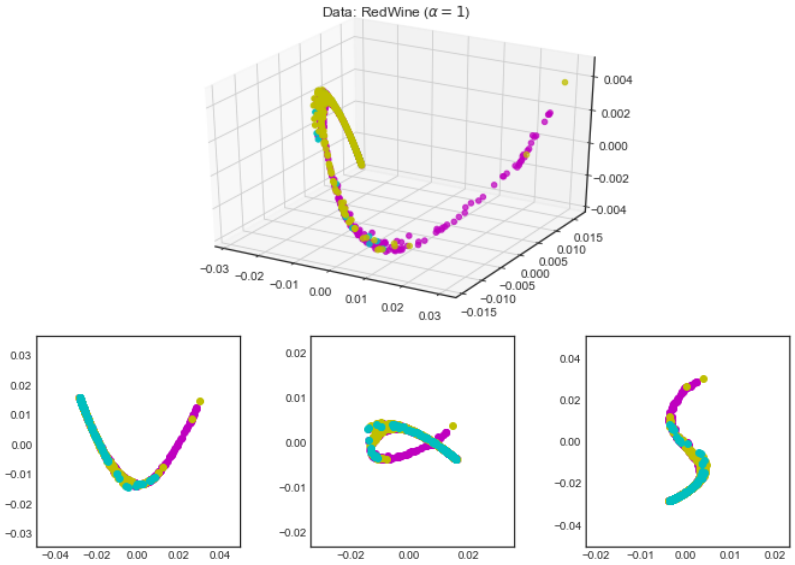

Figure 4.2.11: Embedding for the Red Wine dataset. The parameters are set to $p = 15$ and $t = 16$. Up: a $3D$ plot of the embedding. Down: $2D$ projections for each pair of variables of the embedding. The color code indicates the class.

As we can observe, aiming to solve the Red Wine classification problem in the embedding space would not give good results. The classes are not separated but overlapped. In any case, the shape of the embedding is a smooth three-dimensional curve and it is an interesting example to extend via OOS methods.

4.2.3 Vowel dataset

The Vowel dataset contains information about 11 vowel sounds from the English language. In this problem, the targets are equally distributed. There are 90 examples of each vowel, adding a total of $N = 990$ examples. The dimension of the problem is $m = 10$. There is no need to clean the dataset, as it does not present outliers nor repeated data.

The spectral decay of the eigenvalues of P for the Vowel dataset is very fast, even for small values of σ, as it is shown in Figure 4.2.12. The image contains the spectral decay

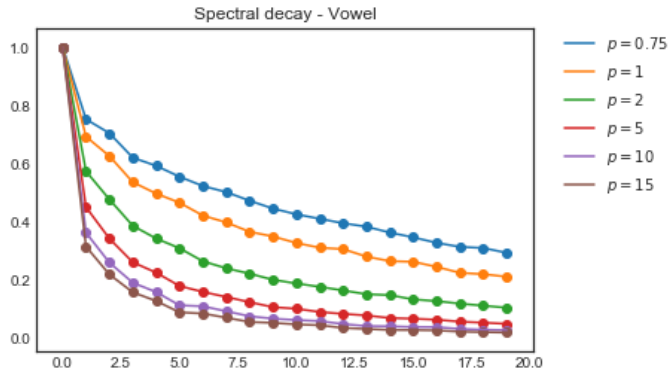

Figure 4.2.12: Spectral decay of the Vowel dataset for different values of σ and $t = 1$. In this case, σ is calculated as the p-th percentile of the distances.

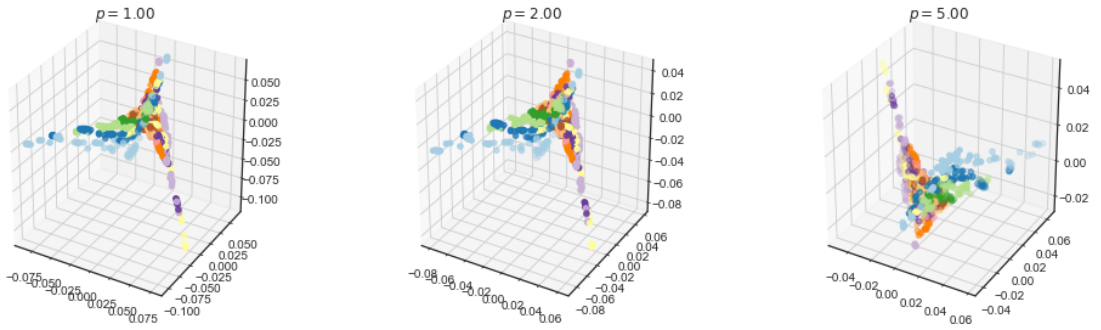

Figure 4.2.13: Embeddings computed for the Vowel dataset for fixed $t = 1$. In this case, σ is calculated as the p-th percentile of the distances.

of the eigenvalues of P for $t = 1$ and different values of p, which, as usual, indicates the percentile of the distances that we are using to set the scale of the kernel. In addition, different 3-dimensional embeddings for this dataset can be seen in Figure 4.2.13. Some separation between classes is achieved setting $p = 1$. Because of that, we will choose this percentile of the distances.

As the spectral decay is very fast in this dataset, the number of steps t we take in the random walk will be small. Figure 4.2.14 contains the spectral decay for different values of t and $p = 1$ fixed. Following the proposed criterion to select the embedding dimension, it is enough to make $t = 8$ steps to select $d = 3$ as the embedding dimension. The final embedding, which will be the one extended in the following sections, is the one shown in Figure 4.2.15.

As a conclusion we can say that certain separation between classes is achieved. Due to the difficulty of the problem, which is composed of $L = 11$ different classes, hoping a perfect separation may be overambitious. In any case, different colors, which represent different targets, are grouped. In addition, the shape of the embedding is peculiar and it is an interesting example to extend via OOS methods.

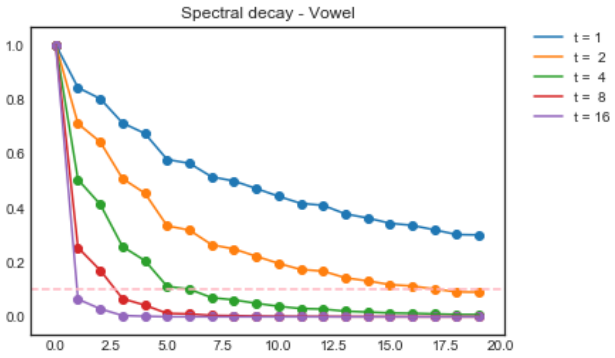

Figure 4.2.14: Spectral decay of the Vowel dataset for different values of t. In this case, σ is calculated as the 1-th percentile of the distances.

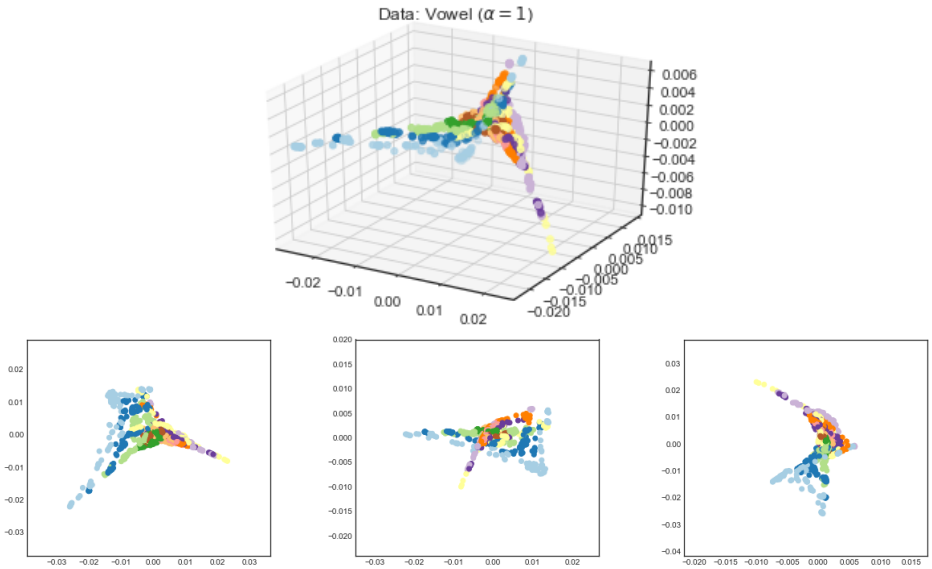

Figure 4.2.15: Embedding for the Vowel dataset. The parameters are set to $p = 1$ and $t = 8$. Up: a $3D$ plot of the embedding. Down: projections for each pair of variables of the embedding. The color code indicates the class.

4.3 Out-of-sample Extension with Nyström's Method

In this Section various experiments will be carried out in order to evaluate the reconstruction quality of the matrix A by extending the embedding to OOS examples using the Nyström's method, as explained in Section 3.2. The embeddings that we will try to extend are the ones obtained in the previous section. The set up is the same for all datasets. Firstly, we select at random the landmark subset, which sometimes will be denoted as training set. Recall that the number of examples composing the entire dataset is denoted by N, the number of patterns in the landmark subset is N_L and we write N' to refer to the number of patterns in the testing set. If we are dealing with a classification problem the proportion of the targets is maintained. Secondly, the embedding for the remaining patterns is computed via Nyström's extension. Finally, the average reconstruction error of A is measured. The whole set up is shown in Figure 4.3.1.

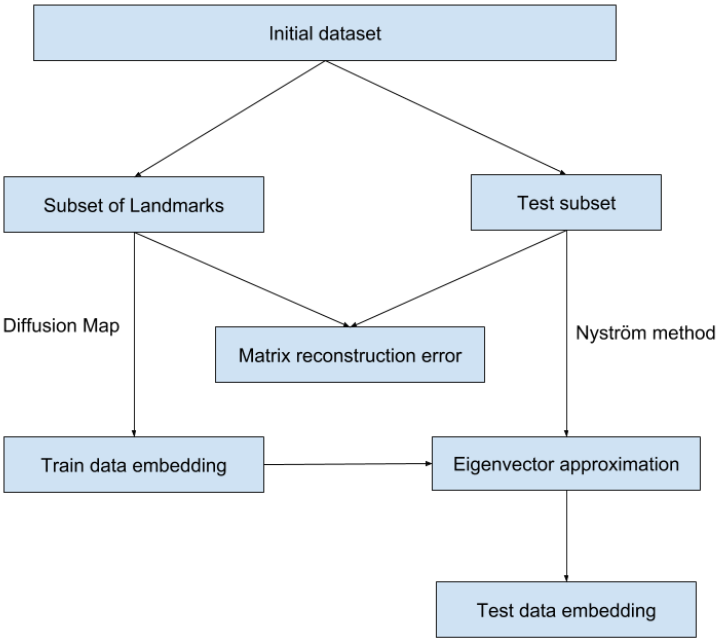

Figure 4.3.1: Graphical representation of the Nyström method to extend the embedding to OOS examples.

Recall that the reconstruction error of the Nyström method was computed in Equation (3.2.2) as

$$ERR_{nys} = \left\| C - B(A^{(L)})^{-1}B^T \right\|_F ,$$

where the meaning of each term can be found in Section 3.2.

4.3.1 Helix OOS with Nyström's Method

The Helix dataset is the first example used to compute the embedding to OOS examples via the Nyström's encoding. Recall that the final embedding was a smooth circle. Both, the configuration and the embedding for the $N = 2000$ patterns via Diffusion Maps, can be seen in Section 4.2.1. As an illustration, Figure 4.3.2 shows the embedding computed for a landmark subset of $N_L = 1000$ patterns selected at random in blue and the extended embedding for the remaining testing patterns in red (right part of the figure). The result is visually accurate, keeping the embedding of the out-of-sample examples in the circle defined by the embedding of the subset of landmarks.

In addition, Figure 4.3.2 (left part of the figure) contains the average reconstruction error of the matrix A as a function of the percentage of patterns picked from the training subset that are considered in the landmark subset. For this experiment, we first split the data into a training subset of size $N_{train} = 1200$ and a testing subset of size $N' = 800$. Then, using a certain percentage p_{train} of patterns picked at random from the training set, which form the subset of landmarks, we compute the reconstruction error for the patterns in the testing set, which are the OOS patterns. This percentage varies in $p_{train} = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, so the number of patterns in the subset of landmarks

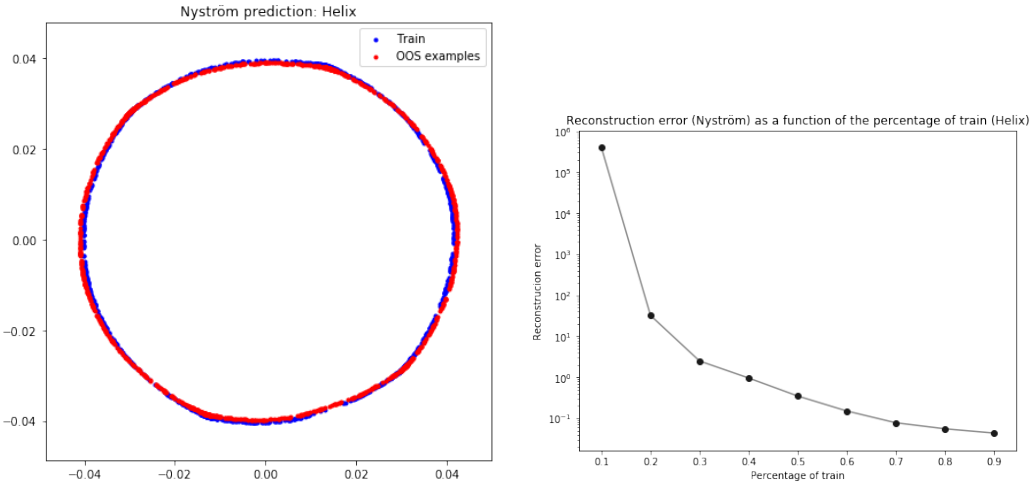

Figure 4.3.2: Left: OOS extension of the Helix dataset for a landmark subset of $N_L = 1000$ patterns chosen at random. The reconstruction error is 0.00917. Right: Average reconstruction error for the Helix dataset when using the Nyström method to calculate OOS extensions for $N' = 800$ patterns in the testing set.

varies in $N_L = \{120, 240, 360, 480, 600, 720, 840, 960, 1080\}$. For each subset of landmarks we redo the computation of $A^{(L)}$, B and C and of the degrees of the points in the testing subset, so only the current patterns are taken into account to measure the reconstruction error. After 10 realizations of each experiment with different train and testing sets, the mean of the error as a function of p_{train} is computed (see Equation (3.2.2)). Note that the size of the testing set is fixed in this experiment. This is because we measure the reconstruction error as the Frobenius norm of a matrix of size $N' \times N'$; therefore, it is sensitive to the testing set size and we cannot compare results from testing sets of different size.

We can observe that, as the train percentage grows, the mean of the reconstruction error after ten realizations of the experiment decreases. Note that the Nyström method would be exact when computing the extension of patterns that are in the subset of landmarks; but this phenomenon never occurs, as we are keeping the testing subset totally independent from the train test. As a conclusion, we can say that the more examples available to train, the more accurate the Nyström approximation is. This is known as consistency.

4.3.2 Red Wine OOS with Nyström's Method

This Subsection contains the experiments carried out to extend the embedding of OOS patterns in the Red Wine dataset via Nyström's encoding. After the cleaning process, in which we delete outliers and repeated patterns, this dataset contains $N = 1175$ examples divided in three different classes. Recall that the embedding we are extending for this example is shown in Figure 4.2.11. Figure 4.3.3 contains the Nyström extension when using half of the patterns ($N_L = 587$) as subset of landmarks and the other half to extend the embedding, always maintaining the proportion of classes in each set.

We can see that the shape of the curve is perfectly imitated. The colors, which encode the classes, are also mapped in similar regions. The reconstruction error of the matrix

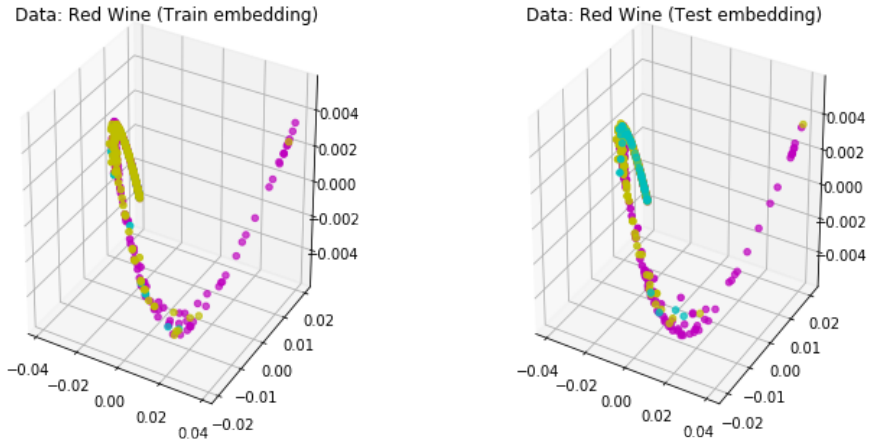

Figure 4.3.3: OOS extension with Nyström's method of the Red Wine dataset for a subset of landmarks of $N_L = 587$ patters chosen at random. Left: embedding for the subset of landmarks. Right: OOS extension for the testing subset.

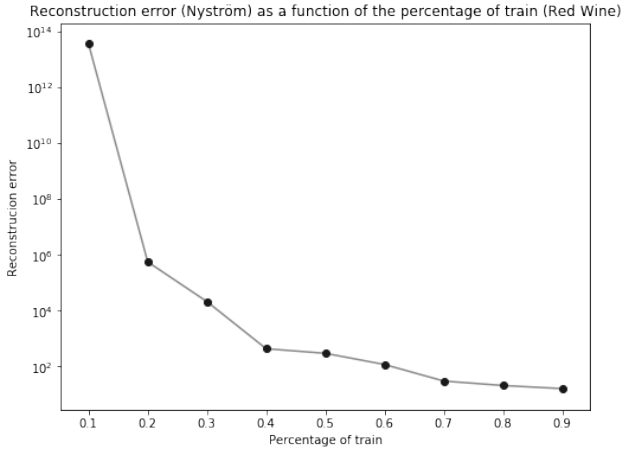

Figure 4.3.4: Average reconstruction error for the Red Wine dataset when using the Nyström method to calculate OOS extensions for $N' = 470$ patterns in the testing set.

A for different percentages of the training set, imitating the approach followed for the Helix dataset, is shown in Figure 4.3.4. The number of OOS patterns, i.e. the number of patterns in the testing set has been set to $N' = 470$ and the remaining patterns are kept in the training set. Then, setting the subset of landmarks as a percentage of patterns of the training set, we reconstruct the matrix A. The experiment is repeated for ten different training and testing subsets, and then the average of the reconstruction error as a function of the landmark size is computed.

4.3.3 Vowel OOS with Nyström's Method

This Subsection contains the experiments carried out to extend the embedding of OOS patterns of the Vowel dataset via the Nyström's encoding. Recall that the embedding we are extending for this example is shown in Figure 4.2.15 and that this dataset did not contain outliers or repeated patterns, so all the examples are taken into account for

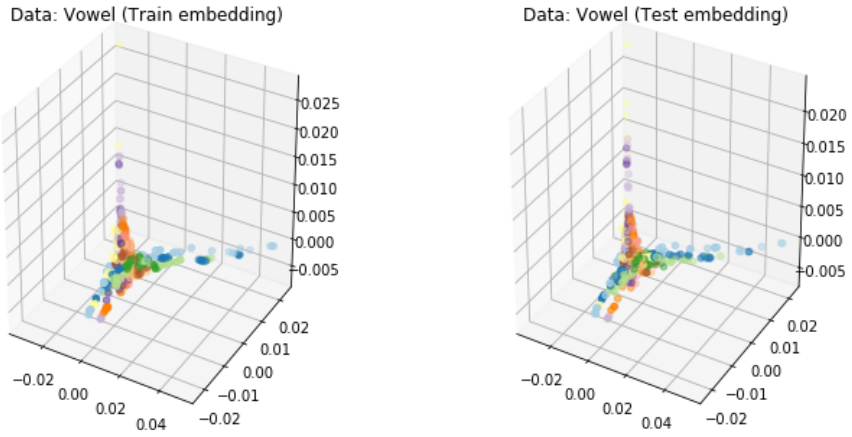

Figure 4.3.5: OOS extension with Nyström's method of the Vowel dataset for a landmark subset of $N_L = 445$ patters chosen at random. Left: embedding for the subset of landmarks. Right: OOS extension for the testing subset.

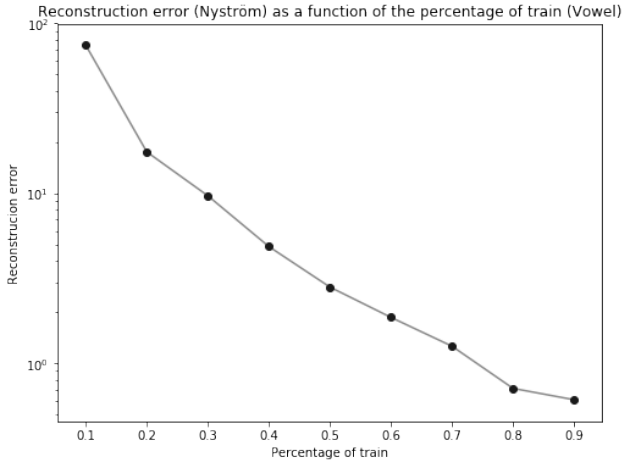

Figure 4.3.6: Average reconstruction error for the Vowel dataset when using the Nyström method to calculate OOS extensions for $N' = 396$ patterns in the testing set.

the computations. In total there are $N = 990$ examples equally distributed in $L = 11$ classes. Figure 4.3.5 contains the Nyström extension when using half of the patterns as subset of landmarks and the other half to extend the embedding ($N_L = N' = 445$), always maintaining the proportion of classes in each set.

We can see that the shape of the embedding is perfectly imitated. The colors, which encode the classes, are also mapped in similar regions. The reconstruction error of the matrix A for different percentages of the training set, following the same approach that the one in the Helix and the Red Wine datasets, is shown in Figure 4.3.6.

The conclusion of this experiment is that, effectively, the reconstruction of the matrix A improves when the number of patterns in the subset of landmarks increases. This phenomenon is observed in the tests performed over the three datasets of this work. Note that the reconstruction error will never be zero in these experiments, since the testing subset is always completely disjoint from the training subset.

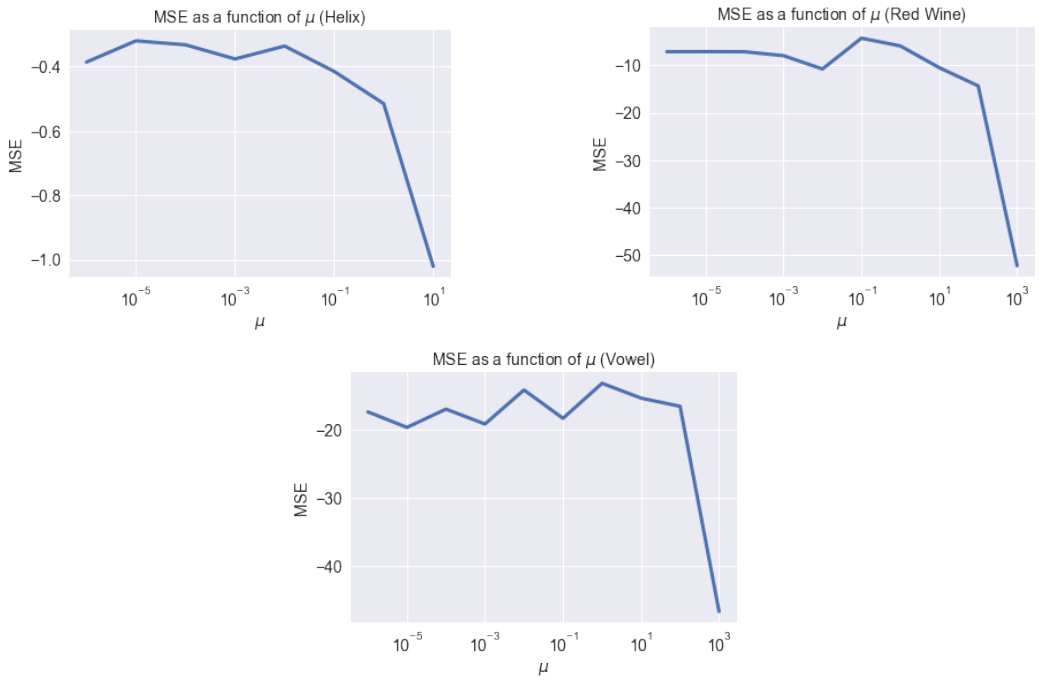

Figure 4.4.1: Grid search for μ. The negative mean squared error (MSE) for each dataset is shown for the values $\mu = \{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 0.1, 1, 10, 100, 1000\}$. The mean is computed averaging in ten folds.

4.4 Out-Of-Sample Extension with Deep Networks

This Section contains experimental results concerning the theory explained in the Section 3.3 of this work. Recall we are going to extend the embedding to OOS patterns training a Multilayer Perceptron (MLP) for regression which we name *encoder*. In this sense, the input of the net is the original point and the target is its embedding, which has been previously computed via Diffusion Maps. The standard loss for an MLP regressor is, as seen in Equation (3.3.1),

$$J^{REG}(\Theta) = \frac{1}{2N}\sum_{i=1}^{N}\|o(x_i,\Theta) - y_i\|^2 + \frac{\mu}{2}\sum_{l=1}^{\mathcal{L}-1}\left\|W^{(l)}\right\|^2 \ ,$$

where Θ is the set of weights, $W^{(l)}$ are the weights of the l-th layer and μ is a parameter to control the importance of the regularization term. The MLPs of the experiments will have $\mathcal{L} = 2$ hidden layers, each one with 100 units or neurons using the $ReLU(z) = max\{0, z\}$ as activation function. In order to select the best value for μ, we perform a grid search cross validation in each dataset. Setting the number of folds to ten, we will measure the average mean squared error for $\mu = \{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 0.1, 1, 10, 100, 1000\}$. Particular results can be seen in Figure 4.4.1, but the procedure is common. First, the dataset is split into a training and a testing subset, which comprise the 40% and 60% of the total of patterns respectively. Then, the training subset is divided in ten folds, which will be used to give the mean square error of each configuration, which is the score used in Figure 4.4.1. The folds and the splits are made preserving the percentage of samples for each class.

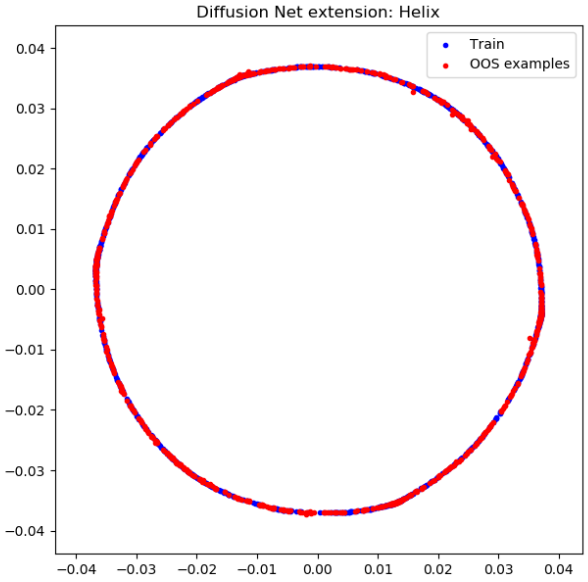

Figure 4.4.2: OOS extension via Diffusion Nets of the Helix dataset for a landmark subset of $N_L = 1200$ patters chosen at random. The regularization parameter is set to $\mu = 10^{-4}$.

4.4.1 Helix OOS with Diffusion Nets

The embedding extension to OOS examples for the Helix dataset via Diffusion Nets has been previously studied in [1]. Replicating the experiments carried out by the authors is complicated, since some details as the regularization parameter μ are not given. In our experiments, the best encoder is obtained setting $\mu = 10^{-4}$, as it can be seen in Figure 4.4.1. The extension of the embedding for the patterns in the testing subset can be seen in Figure 4.4.2. The color code is as follows: blue indicates the embedding computed for the landmark subset and red is the extension computed by the net. Recall that the proposal in [1] was to add a new penalization term with the aim of preserving the eigenvector structure of the Diffusion Map embedding. However, good results are also obtained when training the net just with the standard loss function.

As when computing the extension via Nyström's encoding, both the subset of landmarks and the OOS samples are embedded in a smooth circle. Further work would contain a comparison of both extensions by means of the reconstruction error of the anisotropic matrix A, following a similar approach to the one given in the previous Section of this work and trying to obtain a faithful low-rank approximation of A via Diffusion Nets.

4.4.2 Red Wine OOS with Diffusion Nets

After the cleaning process described in Section 4.2.2, the Red Wine dataset was composed of $N = 1175$ examples. The grid search for the parameter μ is carried out with 705 patterns that are split in ten folds. The best encoder is obtained setting $\mu = 0.1$, as it can be seen in Figure 4.4.1. The OOS extension, which can be observed in Figure 4.4.3 is computed

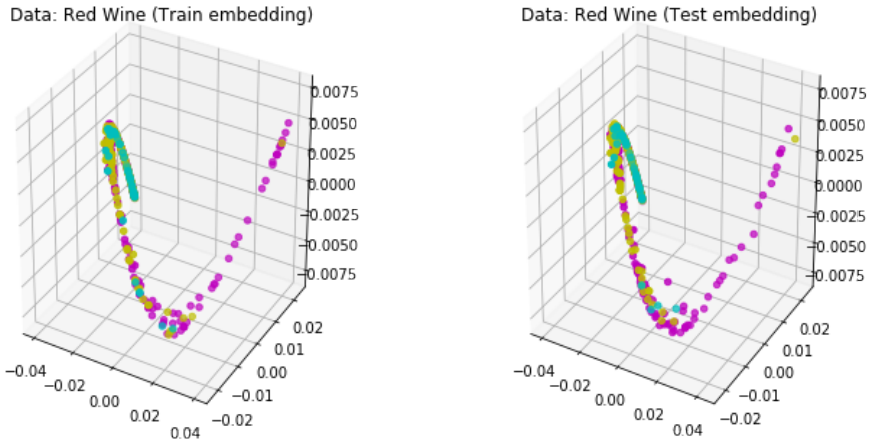

Figure 4.4.3: OOS extension via Diffusion Nets of the Red Wine dataset for a landmark subset of $N_L = 705$ patterns chosen at random. Left: embedding for the subset of landmakrs. Right: OOS extension for the testing subset. The regularization parameter is set to $\mu = 0.1$.

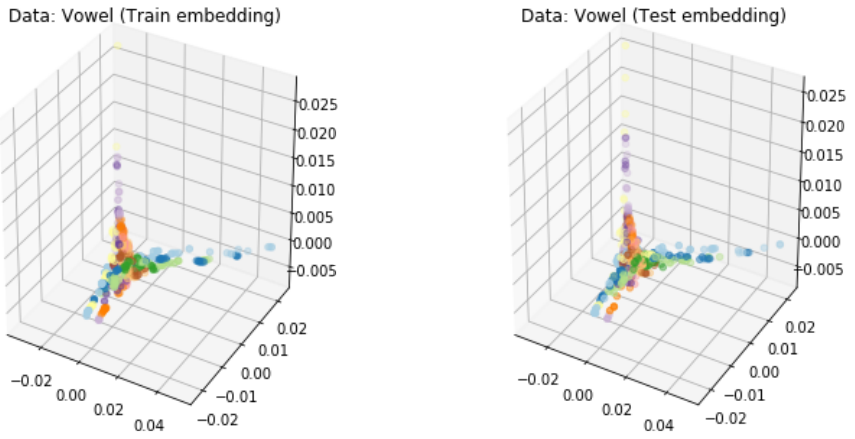

Figure 4.4.4: OOS extension via Diffusion Nets of the Vowel dataset for a landmark subset of $N_L = 594$ patterns chosen at random. Left: embedding for the subset of landmarks. Right: OOS extension for the testing subset. The regularization parameter is set to $\mu = 1$.

for the remaining patterns. As usual, different colors indicate the class belonging. On the left, we can see the embedding computed via Diffusion Maps for the landmark subset and, on the right, the extension given by the net.

The embedding, as with Nyström's encoding, is recovered faithfully. Patterns of the same class are mapped to similar regions in the embedding space. In addition, the shape of the curve is also recovered. Note that both, the extension with the Nyström's method and with the one with Diffusion Nets, are shown using the same training and testing subsets.

4.4.3 Vowel OOS with Diffusion Nets

For the Vowel dataset the same approach has been followed. First, the 60% of the total of the patterns is used for the grid search. The best model is obtained by setting $\mu = 1$, as we can see in Figure 4.4.1. The extension of the embedding for the remaining patterns can be seen in Figure 4.4.4. On the left we can see the embedding computed via Diffusion Maps for the landmark subset and, on the right, the extension given by the encoder. The color code is related to the class belonging and its distribution in the embedding space is kept by the net.

In addition, comparing the extension achieved by the net and the one obtained by the Nyström's method we can see the results are very similar. The training and testing set are the same in both sections, that is one containing the OOS extension with the Nyström's method (see Figure 4.3.5, Section 4.3.3) and the extension given by the net. Both approaches are similar in the sense that the shape of the embedding is well visually imitated, and the color distribution, which indicates the class belonging, is also similar in both OOS methods.

Chapter 5

Discussion and Further Work

The theory of Manifold Learning is important to find the solution of many problems in Machine Learning. In particular, due to its good performance, spectral algorithms, like Laplacian Eigenmaps and Diffusion Maps, are very popular nowadays. Their application is specially interesting in the fields of clustering and dimensionality reduction. However, these methods are computationally expensive, as they require the diagonalization of a square kernel matrix, whose cost is $\mathcal{O}(N^3)$. In addition, in order to give the embedding of a new point, we need to recompute the kernel matrix and redo its eigenanalysis. Different extensions to avoid this situation have been proposed in the literature. In this work we have considered the following two:

- To extend the eigenvectors of the diffusion matrix by means of the Nyström's method, which is a technique usually applied in low-rank approximation of kernel matrices.
- To learn the encoding given by Diffusion Maps with neural networks for regression, that is known as Diffusion Nets.

In addition, Diffusion Maps have other challenges: there are two parameters that need to be set. On the one hand, the kernel scale σ. Our proposal is to choose it by analyzing the matrix of distances computed for the sample and selecting a percentile p, so σ is the p-th percentile of the distances. On the other hand, as these spectral algorithms are related to Markov chains and random walks, we also need to fit the parameter t, which is the number of steps to be taken into account. The selection of these parameters is specially difficult, as there is no clear and well accepted way of comparing embeddings.

It is not clear what the best approach to extend the embedding is. Because of that, this work has been focused on the following main objectives:

1. Computing the embeddings for three different datasets, trying to find the best combination of the parameters σ, p and t.
2. Measuring the low-rank approximation of the kernel matrix when integrating the Nyström's encoding into Diffusion Maps.
3. Training neural networks for regression to extend the embedding. In order to select

the best network, we need to fit the regularization parameter. This analysis has not been done before.

4. Comparing the extension of the embedding to out-of-sample examples when using either Nyström's method or Diffusion Nets.

From the results seen in Section 4 of this work, two important conclusions can be drawn. We summarize them next.

Regarding the low-rank approximation quality, we have observed a common behavior: the low-rank approximation of Diffusion Maps improves, as expected, when the number of examples in the training set increases. In the case of the Helix dataset, which was a synthetic example, the decay of the reconstruction error is very fast. For the Vowel dataset we observe the same phenomenon, but in the case of the Red Wine the reconstruction error does not decay once it has achieved the value 10^2.

Regarding the comparison of the extensions, we have seen that both, Nyström's approach and Diffusion Nets, lead to good results. Both algorithms embed the training and testing data maintaining the shape of the embedding. In addition, when the problem we are dealing with is of classification, points of the same class are mapped in similar regions of the space.

This line of research suggests many possible directions to improve in further work, some of which we are currently undertaking:

- As we have suggested in this work, the percentile of distances p and the number of steps t may be related. Further work would be to study this relation. In addition, the criterion to set the embedding dimension is not clear, as it changes when varying p and t, so this also needs to be studied.

- Due to time limitations, we have only carried out tests with the encoder trying to extend the embedding to out-of-sample examples. However, the decoder is also interesting and trying to replicate the experiments in [1] but giving more information about the network configuration, carrying out a grid search for the regularization parameter, would be part of the future work. Another idea is to test the decoder in real datasets to do data augmentation.

- Even though we have compared the extension of the embeddings given by Diffusion Nets and Nyström's method, we have only done it in a visual way. However, once the Diffusion Net has been trained to extend the embedding, we could obtain also an eigenvector extension (as seen in Section 3.1), so a low-rank approximation of the diffusion matrix is also obtained. Although we have not developed the theory to do that, it is an interesting research line.

- In this work we have seen that using neural networks to extend the embedding leads to good results. In [1] the authors also proposed the implementation of an autoencoder, which would consist in setting the output of the encoder as input of the decoder. This is used for outlier detection. However, we could also train an autoencoder using a multilayer perceptron that tries to replicate the input, which would be the initial dataset. There would be a hidden layer l that tries to imitate the embedding and the loss function would be split in two terms. On the one hand, a term that penalizes

that the values of the layer l are different to the embedding computed by Diffusion Maps. On the other hand, a second term whose aim is to penalize that the input and the output differ.

Bibliography

[1] Gal Mishne, Uri Shaham, Alexander Cloninger, and Israel Cohen. Diffusion nets. *Applied and Computational Harmonic Analysis*, 2017.

[2] Richard Socher and Matthias Hein. Manifold Learning and Dimensionality Reduction with Diffusion Maps. In *Seminar report, Saarland University*, 2008.

[3] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[4] Yoshua Bengio, Jean-françois Paiement, Pascal Vincent, Olivier Delalleau, Nicolas L Roux, and Marie Ouimet. Out-of-sample extensions for LLE, Isomap, MDS, eigenmaps, and Spectral Clustering. In *Advances in neural information processing systems*, pages 177–184, 2004.

[5] Ulrike von Luxburg. A Tutorial on Spectral Clustering. *CoRR*, abs/0711.0189, 2007.

[6] Ronald R Coifman and Stéphane Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.

[7] Christopher K. I. Williams and Matthias Seeger. Using the Nyström Method to Speed Up Kernel Machines. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press, 2001.

[8] Nasser M Nasrabadi. Pattern recognition and machine learning. *Journal of electronic imaging*, 16(4):049901, 2007.

[9] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[10] John C Gower and Gavin JS Ross. Minimum spanning trees and single linkage cluster analysis. *Applied statistics*, pages 54–64, 1969.

[11] Ronald R Coifman and Matthew J Hirn. Diffusion maps for changing data. *Applied and computational harmonic analysis*, 36(1):79–107, 2014.

[12] Mohamed-Ali Belabbas and Patrick J Wolfe. Spectral methods in machine learning and new strategies for very large datasets. *Proceedings of the National Academy of Sciences*, 106(2):369–374, 2009.

[13] Alex Kulesza, Ben Taskar, et al. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, 2012.

[14] Aren Jansen, Gregory Sell, and Vince Lyzinski. Scalable out-of-sample extension of graph embeddings using deep neural networks. *Pattern Recognition Letters*, 94:1–6, 2017.

[15] James McQueen, Marina Meilă, Jacob VanderPlas, and Zhongyue Zhang. Megaman: Scalable manifold learning in Python. *The Journal of Machine Learning Research*, 17(1):5176–5180, 2016.

[16] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

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

[18] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[19] Mahesan Niranjan and Frank Fallside. Neural networks and radial basis functions in classifying static speech patterns. *Computer Speech & Language*, 4(3):275–289, 1990.