Design and HPC Implementation of Unsupervised Kernel Methods in the Context of Molecular Dynamics

PhD Thesis for the Course in:
Science and Technology for Electronic and Telecommunication Engineering

Marco Jacopo Ferrarotti

supervised by
Dr. Walter Rocchia
Prof. Rodolfo Zunino

UNIVERSITÀ DEGLI STUDI DI GENOVA

iit ISTITUTO ITALIANO DI TECNOLOGIA

February 28, 2018
Genova, Italy

Abstract

The thesis represents an extensive research in the multidisciplinary domain formed by the cross contamination of Unsupervised Learning and Molecular Dynamics, two research fields that are coming close creating a breeding ground for valuable new concepts and methods.
In this context, at first, we describe a novel engine to perform large scale kernel k-means clustering. We introduce a two-fold approximation strategy to minimize the kernel k-means cost function in which the trade-off between accuracy and execution time is automatically ruled by the available system memory. Moreover, we define an effective parallelization scheme well suited for GPU endowed state-of-the-art parallel architectures.
We prove the effectiveness of the method testing a working MPI - CUDA implementation on standard Machine Learning datasets and on an Molecular Dynamics real-case application scenario.
Secondly, we introduce the concept of principal paths in data space. Those paths can be interpreted as local Principal Curves and in the statistical mechanics realm correspond to, possibly Minimum, Free Energy Paths. Here we move that concept from physics to data space and derive a regularized k-means algorithm to compute them in the original and kernel space. In this fully unsupervised environment, we successfully apply the Bayesian framework of evidence maximization to perform in-sample model selection on the introduced regularization parameter.
We apply the method to standard Machine Learning datasets, dynamical systems and in particular on Molecular Dynamics trajectories showing the generality, the usefulness of the approach and its superiority with respect to other related techniques.

Contents

Part I

Introduction

Chapter 1

Scope of the Thesis

1.1 From simulations to human interpretable models

Molecular Dynamics (MD) is a computational technique that allows the dynamic of a system to be followed at atomistic resolution, therefore representing a virtual microscope to investigate chemical reactions and transitions in molecular systems of interest. With this respect, for example, MD is currently the tool of choice for the in-silico study of dynamical protein-ligand binding [1, 2, 3].

Unsupervised Learning (UL) is that area of Statistical Learning devoted to learn *non trivial* representations of the data starting from unlabeled samples. As such, it is a cognitively difficult problem where a machine is asked to infer the underlying structure of the data. Examples of methods in this area being clustering, manifold learning and dimensionality reduction with a wide spectrum of application domains.

This two apparently distant research fields recently found a point of contact in the necessity to automatically process and extract useful information out of long MD simulations. With *state of the art* High Performance Computing (HPC), now endowed with general purpose GPUs (gpGPUs), MD trajectories with up to 10^{10} frames (i.e. microsecond long trajectories) are now a reality. Their analysis can be a daunting task from the standpoint both of the human intervention time and of the necessary computation and storage requirements. Machine Learning (ML) techniques in general, and more specifically large scale UL techniques, with their ability to learn compact, meaningful representations of the data are valuable tools in this context. A successful example of this being found in Markov State Model (MSM) [4] which rely to various extent on clustering algorithms in order to infer a coarse grained, human interpretable kinetic model starting from MD data.

The resulting multidisciplinary field is rich in both scientific and technological challenges. Indeed from a technological standpoint the urge of deploying automatic processing tools on the same HPC facilities where MD simulations are running, together with the special requirements of MD data, as it will be discussed in chapter 2, pushes the development of highly efficient UL techniques. From a scientific point of view, not only UL applied to MD can help in better understanding biomolecular processes of interest; but MD can also inspire totally new learning concepts and algorithms resulting in significant contributions to the UL field as it will be discussed in chapter 3.

The work presented here is naturally set in this cross contaminating domain, two will be the main objectives of the thesis:

1. The design of an efficient kernel k-means engine to perform large scale clustering on gpGPU endowed HPC facilities (with particular attention to MD trajectories as application scenario).
2. The theoretical derivation of a kernel algorithm to find Principal Paths in data space, a new cognitively sound learning problem inspired by MD.

The motivations behind those two objectives will be detailed respectively in chapter 2 and chapter 3, whereas in the following a brief overview of MD and UL is given.

1.2 Molecular Dynamics

Hereafter we are going to give a brief overview of MD aiming at introducing those concepts that will be relevant for later discussions.

Let us consider a molecular system of N_a atoms, its micro-state will be then identified by $\underline{x}$ and $\underline{p}$ (i.e. the $3N_a$ dimensional vectors of positions and momenta respectively). According to classical mechanics the total energy of the system subject to a given potential $U(\underline{x})$ is given by the Hamiltonian:

$$H(\underline{x}, \underline{p}) = U(\underline{x}) + \frac{\underline{p}^2}{2m} \tag{1.1}$$

The time evolution of such system is governed by:

$$\frac{d\underline{x}}{dt} = \frac{\partial H}{\partial \underline{p}} \tag{1.2}$$

$$\frac{d\underline{p}}{dt} = -\frac{\partial H}{\partial \underline{x}} \tag{1.3}$$

In its most basic formulation MD is a computational technique that starts from an initial state of the system $(\underline{x}^0, \underline{p}^0)$ and numerically integrates Eq.1.2 with a fixed time-step Δt for an empirical potential $U(\underline{x})$ (i.e. a potential energy properly designed and parametrized to model the bonded and non-bonded interactions among the atoms in the system). The output of such simulation therefore will be in the form of a sequence of N conformational frames, namely $\{\underline{x}^0, \underline{x}^{\Delta t}, \underline{x}^{2\Delta t}, \ldots, \underline{x}^{N\Delta t}\}$. In this sense MD is commonly viewed as a virtual microscope that allows one to closely follow the time evolution of a process with atomistic resolution [5].

Simulating a system at constant temperature

In general terms, it is interesting to simulate a system at constant temperature, therefore in a real application scenario the numerical integration of Eq.1.2, which would normally conserve the total energy, is paired with a thermostat that allows energy fluctuations. The Andersen technique [6] is probably the simplest example of thermostat where thermalization is achieved by drawing the momentum of randomly selected particles from the equilibrium distribution $e^{-\beta\frac{\underline{p}^2}{2m}}$.

Several other techniques are available in the literature (see e.g. [7, 8]) and one should keep in mind that simulating the time evolution of a thermalized system is a non trivial problem [9] which relies on central concepts in statistical physics such as the one of ergodicity [10]. We will not enter here into the details of such discussion, limiting ourselves to say that when a thermostat is applied to the system then an MD simulation can be thought as a process where the phase space is explored by means of thermal fluctuations. In such scenario the probability for the system of being found in a state $(\underline{x}, \underline{p})$ is given by the well known Boltzmann distribution:

$$P(\underline{x}, \underline{p}) \propto e^{-\beta H(\underline{x},\underline{p})} = e^{-\beta U(\underline{x})} e^{-\beta\frac{\underline{p}^2}{2m}} \tag{1.4}$$

Such distribution is of paramount importance providing a connection between the thermodynamics of macroscopic states and the statistics of microscopic system conformations. For example, having defined the occupancy probability of a given macro-state A as:

$$P_A \propto \int_A d\underline{x}\; d\underline{p}\; e^{-\beta H(\underline{x},\underline{p})} \tag{1.5}$$

then the free energy $\mathcal{F}_A$ can be defined as:

$$\mathcal{F}_A = -k_B T \log P_A \tag{1.6}$$

Reconstructing the Free Energy Surface

It is worth observing that biomolecular processes of interest usually evolves through a series of metastable states corresponding to local minima of the underlying free energy $\mathcal{F}$ just introduced.

The first step for studying this kind of process via an MD simulation is to define a small set of Collective Variables (CVs) meant to characterize the process i.e. a set of reaction coordinates that one can monitor to clearly identify transitions among metastable states. A CV is defined as a given function $\theta(\underline{x})$ of the system coordinates. It can be as simple as a dihedral angle, even though in real application scenarios more sophisticated and computationally expensive descriptors are often used (see e.g. [11]).

Let us assume, for the sake of simplicity, to define a single CV $\theta(\underline{x})$. Now for a given state described by $z = \theta(\underline{x})$, the occupancy probability can be defined as:

$$P(z) \propto \int d\underline{x} e^{-\beta U(\underline{x})} \delta(z - \theta(\underline{x})) \tag{1.7}$$

and a Free Energy Surface (FES) with respect to the CV can be computed as:

$$\mathcal{F}(z) = -k_B T \log P(z) + A \tag{1.8}$$

With a thermalized MD simulation we would like to estimate the probability distribution $P(z)$ in order to reconstruct the FES $\mathcal{F}(z)$ from which a series of relevant information can be extracted e.g. the stability of the states or the tranistion rates among them. However as it is known from Arrhenius law [12], the probability of escaping a free energy minimum by thermal fluctuations is exponentially small with respect to the height of the barrier. Therefore chances are that, being the simulation time finite, if the system is initialized in a stable state corresponding to a free energy minimum it will remain confined there.

This problem is usually reffered to as the problem of rare events and several accelerated sampling techniques have been proposed in the literature in order to partially overcome it (see e.g. metadynamics [13, 14]). The simplest example of such techniques (i.e. umbrella sampling [15]) can be easily understood also in this context, indeed let us suppose to introduce an additive artificial potential depending only on the CV i.e. $V(\theta(\underline{x}))$. Then performing a thermalized MD simulation with such bias potential would correspond to sampling the following probability of states:

$$\hat{P}(z) \propto \int d\underline{x} e^{-\beta(U(\underline{x})+V(\theta(\underline{x}))} \delta(z - \theta(\underline{x})) = e^{-\beta V(z)} P(z) \tag{1.9}$$

which in turns defines the following FES:

$$\hat{\mathcal{F}}(z) = -k_B T \log \hat{P}(z) = \mathcal{F}(z) + V(z) + A \tag{1.10}$$

The introduced artificial potential has an additive effect on the FES $F(z)$ and as such can be designed to compensate the orginal energy barriers thus providing a better sampling of the phase space.

We close this section stating that MD simulations, when paired with adequate accelerated sampling techniques, allow the study of a molecular process by means of FES reconstruction [16, 17] and can be viewed as a source of conformational frames properly sampled from the underlying micro-state occupancy probability distribution.

1.3 Unsupervised Learning

The UL problem is usually defined in relation to its supervised counterpart as the problem of learning from data *without labels*. Indeed in the supervised learning paradigm a Learning Machine is presented with a set of sample-label pairs $(\underline{x}_i, y_i)$ where samples are drawn from $P(x)$ (Generator) and the labels are drawn from $P(y|x)$ (Supervisor). In this context the goal of learning is naturally identified with the one of selecting among a given set of learnable functions the one that best approximates the supervisor's response i.e. minimize the expected value $E_{P(\underline{x},y)}[\Omega(f(\underline{x}), y)]$ of a given loss function $\Omega(f(\underline{x}), y)$. The theoretical foundations of such problem are solid and have to be ascribed mainly to the work of Vapnik [18] who introduced several concepts of paramount importance such as the one VC entropy and VC dimension (measures related to the generalization ability of a set of functions) and the concept of Strucutral Risk Minimization i.e. a consistent learning principle where the generalization ability of the function learned is made a controlling variable of the optimization process.

When the Supervisor (i.e. set of labels) is removed from the paradigm one has to redefine the objective of the learning process. While this may still be an open question, it is common in the literature to identify UL with regularized manifold learning [19] and topological data analysis [20]. More intuitively, given a set of data $\mathbf{X}$, the problem of learning from those data without labels can be understood as the problem of learning a *non trivial representation* $\mathbf{W}$. In such scenario the learned representation has somehow to be evaluated with respect to the samples themselves as shown in Fig.1.1.

Both the UL problems treated in this dissertation, namely clustering and Principal Curve (PC) analysis are included in this intuitive definition. Indeed in the first case one aims at representing the data as a small set of prototypes whereas in the second case one aims at representing the data with a continuous one-dimensional curve.

Regularized functionals

The concept of *non trivial representation* relates with the one of regularization, a central aspect in the statistical learning theory [21, 22, 23]. Indeed directly minimizing an empirical error without constraints on the set of learnable representations may easily lead to overfitting. It is worth observing that a regularization can be either implicit (as in the case of clustering N samples into N_C clusters where one set $N_C << N$) or explicit. For instance,

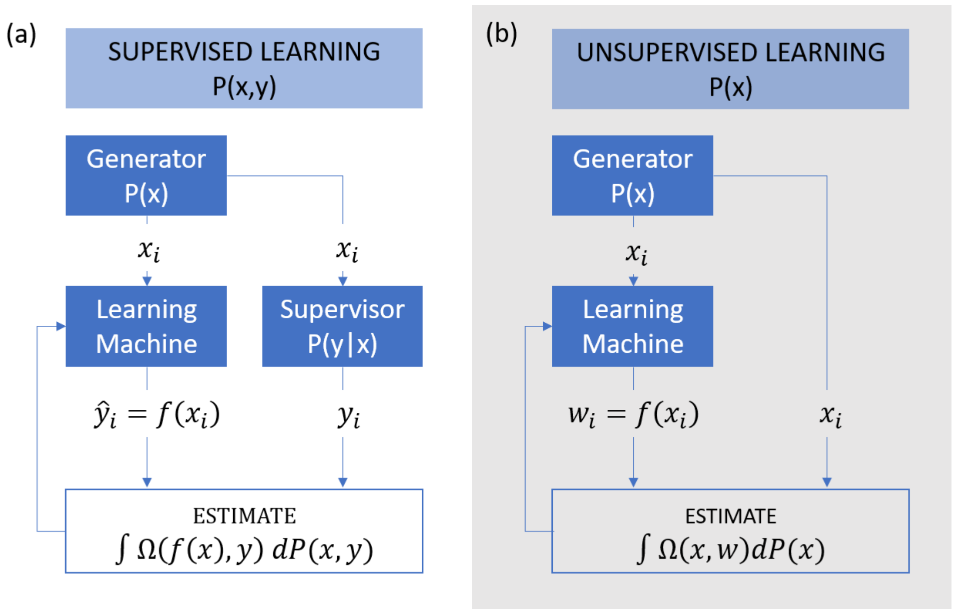

Figure 1.1: (a) Supervised learning paradigm where a Learning Machine is paired with a supervisor providing labels y_i for each training sample x_i. The learning procedure is described as the problem of minimizing the expected loss starting from the training samples. (b) UL paradigm where a Learning Machine is fed with unlabelled samples. In this context one may speculate that a good learning procedure is the one that minimizes the expected representation error. For example, as explained in the main text, one can minimize a regularized empirical representation error.

one could formulate a learning problem as:

$$\min_{\mathbf{W}} \gamma \Omega_X(\mathbf{X}, \mathbf{W}) + \lambda \Omega_W(\mathbf{W}) \tag{1.11}$$

where Ω_X is the empirical error of the representation $\mathbf{W}$ on the data $\mathbf{X}$ and Ω_W is a penalty term penalizing the complexity of the representation. The problem of properly setting the trade-off between representation error and regularization (i.e. properly setting the values of γ and λ) is the generally difficult problem of model selection, an essential part of the learning process.

Bayesian evidence

It is worth noting that one may as well look at the regularized learning problem described by Eq.1.11 as a maximum posterior problem in the framework of Bayesian inference. Indeed, simply taking the negative exponential of the regularized cost and assuming a proper normalization, the following probability can be defined:

$$\begin{aligned} P(\mathbf{W}|\mathbf{X},\gamma,\lambda) &= \frac{e^{-\Omega(\mathbf{W},\mathbf{X},\gamma,\lambda)}}{Z(\gamma,\lambda)} \\ &= \frac{e^{-\gamma\Omega_X(\mathbf{W},\mathbf{X})}}{Z_X(\gamma)} \frac{e^{-\lambda\Omega_W(\mathbf{W})}}{Z_W(\lambda)} \frac{Z_X(\gamma)Z_W(\lambda)}{Z(\gamma,\lambda)} \end{aligned} \tag{1.12}$$

where Z, Z_X and Z_W are normalizing constants defined as:

$$Z(\gamma,\lambda) = \int e^{-\Omega(\mathbf{W},\mathbf{X},\gamma,\lambda)} dW$$

$$Z_D(\gamma) = \int e^{-\gamma\Omega_X(\mathbf{W},\mathbf{X})} dW$$

$$Z_W(\lambda) = \int e^{-\lambda\Omega_W(\mathbf{W})} dW$$

Comparing Eq.1.12 with the well known Bayes theorem for conditional probability:

$$P(\mathbf{W}|\mathbf{X},\gamma,\lambda) = P(\mathbf{X}|\mathbf{W},\gamma,\lambda)P(\mathbf{W}|\gamma,\lambda)\frac{1}{P(\mathbf{X}|\gamma,\lambda)}$$

one can obtain the following definition for the likelihood of the data:

$$L(\gamma,\mathbf{W}) = P(\mathbf{X}|\mathbf{W},\gamma,\lambda) = \frac{e^{-\gamma\Omega_X(\mathbf{W},\mathbf{X})}}{Z_X(\gamma)} \tag{1.13}$$

for the prior probability of the model:

$$P(\mathbf{W},\lambda) = \frac{e^{-\lambda\Omega_W(\mathbf{W})}}{Z_W(\lambda)} \tag{1.14}$$

and for the evidence:

$$E(\gamma,\lambda) = P(\mathbf{X}|\gamma,\lambda) = \frac{Z(\gamma,\lambda)}{Z_X(\gamma)Z_W(\lambda)} \tag{1.15}$$

.

At this point the learning process can be formulated as the following two level inference procedure:

- 1st Level: starting from a given hypothesis (γ,λ) infer the best model $\mathbf{W}_{MP}$ through a maximum posterior criterion (i.e. minimizing the regularized cost function).
- 2nd Level: Infer the best parameters (γ,λ) with a maximum posterior criterion on the hypothesis set. Assuming a flat prior probability (i.e. stopping the inference at this second level) such maximum posterior criteria is equivalent to a maximum evidence criteria as shown by:

$$\begin{aligned}(\gamma, \lambda)_{\mathrm{ME}} &= \max_{\gamma,\lambda} P(\gamma, \lambda|\mathbf{X}) \\ &= \max_{\gamma,\lambda} \frac{P(\mathbf{X}|\gamma, \lambda)P(\gamma, \lambda)}{P(\mathbf{X})} \\ &= \max_{\gamma,\lambda} E(\gamma, \lambda)\end{aligned}$$

The Bayesian inference perspective on learning therefore is valuable, providing a theoretical framework for the model selection of the parameters as extensively discussed in [24].

1.4 Outline of the thesis

The rest of the thesis is structured as follow: chapter 2 and chapter 3 complete the introductory part giving the motivations behind the two main objectives of the thesis and framing them into the relevant literature. More specifically chapter 2 introduces the problem of clustering MD trajectories and chapter 3 deals with the problem of finding Principal Paths in data space. Three methodological chapters follows where the original contribution of the thesis is detailed. Chapter 4 describes the novel Distributed Kernel K-means (DKK) clustering engine where a two-fold approximation is paired with an efficient distribution strategy to tackle the computational burden of standard kernel k-means. Chapter 5 then shows how an acceleration strategy can be effectively designed for DKK in order to cluster MD trajectories on HPC facilities where CPUs are paired to accelerators, with particular attention to gpGPUs for which an efficient CUDA implementation is proposed. At last, chapter 6 describes a regularized kernel k-means functional that can be minimized in order to find principal paths in data space together with an actual optimization algorithm and the derivation of a maximal evidence principle for in-sample model selection of the parameters. Two experimental chapters, namely chapter 7 and chapter 8 follows, where the developed methods are validate against toy models, standard datasets in the ML literature and a real MD appliction scenario. Overall conclusions close the manuscript.

Chapter 2

Unsupervised Learning Applied to Molecular Dynamics

Hereafter a first connection between Molecular Dynamics (MD) and Unsupervised Learning (UL) is described. More precisely, we will discuss how clustering methods can be applied in the automatic analysis of MD trajectories.

At first clustering is framed in the context of UL presenting two of the most widely known clustering techniques i.e k-means algorithm (both in its linear and kernelized formulation) and the related k-medoids algorithm. Secondly, we discuss how clustering has been successfully used in the literature in order to build coarse grained models inferred from MD data.

At last we close this chapter identifying kernel k-means as a valuable clustering algorithm in this domain and accordingly, we define the first objective of the thesis to be the development of an efficient large scale kernel k-means algorithm.

2.1 Clustering data

The problem of clustering can be informally described as the problem of *partitioning unlabeled data samples into meaningful groups.* Since 1967, when k-means was originally introduced [25], a variety of different clustering algorithms arose without a clear all-around winner. Reasons behind such a fragmented panorama have to be found in the ambiguity of what a *meaningful cluster* is. Different cluster definitions indeed induce different grouping strategies e.g density based definitions lead to algorithms such as DBSCAN [26] whereas definitions based on the spectral property of a similarity matrix leads to Spectral Clustering [27]. Even though desirable, a unified theory of clustering seems far from being achieved. Recent developments in such direction have to be found in the work of Kleinberg [28] who, starting from a small set of reasonable axioms for a clustering function proved an *impossibilty* theorem and in the works by Shai Ben-David [29, 30] who went one step further moving the attention from clustering functions to clustering quality measures proving how in such domain a working set of axioms can be found.

Hereafter, for the sake of simplicity, the clustering problem will be identified with what is usually found in the literature as vector quantization i.e. the problem of learning a discrete representation for the data in the form of N_C prototype vectors $\underline{w}_i$.

2.1.1 K-means Algorithm

Let us consider a set $\mathbf{X}$ of N data samples $\underline{x}_i \in \mathbb{R}^d$ and let us define a partition of the data in the form of labels $u_i \in [1, N_C]$. One can now define the following quantization error to be minimized:

$$\Omega(\mathbf{X}, \mathbf{W}) = \sum_{i=1}^{N} \sum_{j=1}^{N_C} ||\underline{x}_i - \underline{w}_j||^2 \delta(u_i, j) \tag{2.1}$$

Finding a global minimum for such non-convex cost is a computationally unfeasible task. The k-means algorithm [31] finds sub-optimal minima starting from an intial set of prototypes $\mathbf{W}^0$ with the following two steps EM-like procedure [32]:

1. assuming a set $\mathbf{W}^t$ of prototypes at a given iteration t, set the labels to be equal to the one minimizing $\Omega(\mathbf{X}, \mathbf{W}^t)$ i.e.

$$u_i^{t+1} = \arg\min_j ||\underline{x}_i - \underline{w}_j^t||^2 \tag{2.2}$$

2. having computed the update labels u_i^{t+1}, minimize Ω with respect to the set of prototypes $\mathbf{W}$ keeping u_i^{t+1} fixed i.e

$$\underline{w}_j^{t+1} = \frac{1}{|\underline{w}_j^t|} \sum_{i=1}^{N} \underline{x}_i \delta(u_i^{t+1}, j) \tag{2.3}$$

As shown in [33], this kind of procedure almost surely converges to a local minimum eventually reaching the stopping condition $u_i^{t+1} = u_i^t,\ \forall i \in [1, N]$ i.e. $P(\lim_{t\to\infty}\{u_i^{t+1} = u_i^t\ \forall i \in [1, N]\}) = 1$. The complexity of the algorithm is $O(NN_C dT)$ where T is the number of iterations needed to converge.

The success of k-means algorithm has to be found mainly in its simplicity and in the clear geometrical interpretation of its results. It is worth observing however that the applicability of k-means as discussed above is limited to those domains where an explicit feature space is known i.e. where one can evaluate Eq.2.3 to compute the explicit coordinates of the centroids. This

Polynomial kernel	$K_{m,n} = (\gamma \underline{x}_m^T \underline{x}_n + c)^l$
Gaussian (RBF) kernel	$K_{m,n} = \exp(-\frac{D^2(\underline{x}_m, \underline{x}_n)}{2\sigma^2})$
Sigmoid kernel	$K_{m,n} = \tanh(\gamma \underline{x}_m^T \underline{x}_n + c)^l$

Table 2.1: Popular kernel functions

may not always be the case, an example being all those situations in which graph-like structured data has to be analyzed. In such scenarios usually one does not have a readily available feature space but rather a similarity or a distance measure. Moreover k-means clustering does not allow non linearities in the data, always looking for linearly separable clusters. To deal with both problems as we are going to show, a kernel extension of the algorithm was proposed in [34].

2.1.2 Kernel K-means Algorithm

The kernel k-means algorithm can be easily derived starting from the two step EM-like procedure proposed before. Let us substitute Eq.2.3 into Eq.2.2 in order to obtain the following self consistent update equation for the set of labels:

$$u_i^{t+1} = \arg\min_j \frac{1}{|w_j^t|^2} \sum_{m,n} \langle \underline{x}_m, \underline{x}_n \rangle \delta(u_m^t, j) \delta(u_n^t, j) - \frac{2}{|w_j^t|} \sum_m \langle \underline{x}_i, \underline{x}_m \rangle \delta(u_m^t, j) \tag{2.4}$$

We can now obtain a kernel version for the k-means algorithm by means of what is usually referred to as *kernel trick* [35] i.e. replacing the inner product among data samples $\langle \underline{x}_m, \underline{x}_n \rangle$ with a generic Mercer kernel function $K(\underline{x}_m, \underline{x}_n)$ i.e.

$$u_i^{t+1} = \arg\min_j \frac{1}{|w_j^t|^2} \sum_{m,n} \mathbf{K}_{m,n} \delta(u_m^t, j) \delta(u_n^t, j) - \frac{2}{|w_j^t|} \sum_m \mathbf{K}_{i,m} \delta(u_m^t, j) \tag{2.5}$$

Several choices are possible as listed in table 2.1 and it is worth observing that if the kernel function depends only by the distance among samples $D(\underline{x}_m, \underline{x}_n)$ then the set of labels can be updated till convergence requiring just the $N \times N$ distance matrix $\mathbf{D}$ thus enabling the algorithm to run also on those structured data where an explicit vector space may not be readily available.

Substituting the inner product with a Mercer kernel function is a legitimate operation that leads to a meaningful kernel k-means algorithm since those kind of functions are proved to be inner product in a possibly unknown transformed space i.e. $K(\underline{x}_m, \underline{x}_n) = \langle \phi(\underline{x}_m), \phi(\underline{x}_n) \rangle$, $\phi(\cdot) : \mathbb{R}^d \to \mathbb{R}^{d'}$.

Iterating Eq.2.5 we are therefore implicitly minimizing the following quantization error with respect to the set of prototypes $\underline{w}_i \in \mathbb{R}^{d'}$ in the unknown transformed space:

$$\Omega(\mathbf{X}, \mathbf{W}) = \sum_{i=1}^{N} \sum_{j=1}^{N_C} ||\phi(\underline{x}_i) - \underline{w}_j||^2 \delta(u_i, j) \tag{2.6}$$

It is worth observing that since the transformation $\phi(\cdot) : \mathbb{R}^d \to \mathbb{R}^{d'}$ is usually unknown the algorithm in the proposed formulation does not give access directly to the set of prototypes $\mathbf{W}$. One however can easily find the prototypes medoids i.e. those samples that in the transformed space are closest to the prototypes:

$$\begin{aligned} \phi^{-1}(\underline{w}_j) \approx \underline{m}_j &= \arg\min_{\underline{x}_l \in \mathbf{X}} \|\phi(\underline{x}_l) - \underline{w}_j\|^2 \\ &= \arg\min_{\underline{x}_l \in \mathbf{X}} \mathbf{K}_{l,l} - 2\frac{1}{|w_j|} \sum_i \mathbf{K}_{i,l} \delta(u_i, j) \end{aligned} \tag{2.7}$$

Approximate kernel k-means

The major shortcoming of kernel k-means has to be found in the intrinsic quadratic complexity of the algorithm due to the kernel matrix evaluation step which limits the applicability of the method to reasonably small datasets. Chitta et al. [36] recently proposed an approximate version of the algorithm in order to reduce such burden based on a centroids sparse representation. By construction, at each iteration of the exact kernel k-means algorithm one is implicitly representing the centroids in the transformed space as a linear combination of the entire dataset:

$$\underline{w}_j^{t+1} = \frac{1}{|\underline{w}_j^t|} \sum_{i=1}^{N} \phi(\underline{x}_i) \delta(u_i^{t+1}, j) \tag{2.8}$$

If one restrict such representation to a sub space spanned by a small set of landmarks $\mathbf{L} \subset \mathbf{X}$ then is easy to demonstrate that the evaluation of $K(\underline{x}_m, \underline{x}_n)\ \forall \underline{x}_m \in \mathbf{X}\ \forall \underline{x}_n \in \mathbf{L}$ is sufficient to iterate the algorithm until convergence i.e. the complexity of the kernel matrix evaluation step is reduced to $O(|L|N)$.

K-means as a limiting case

We close this section showing that, interestingly enough, standard k-means can be obtained as a limiting case of kernel k-means for $\sigma \to \infty$ when a gaussian kernel is used. One may argue that using kernel k-means to emulate k-means results is a rather inefficient way to proceed however this may be a viable solution to perform standard k-means clustering on datasets where an explicit feature space is not available.

Let us start expanding the gaussian kernel function with euclidean distance around 0:

$$K_{m,n} = \lim_{\sigma\to\infty} \exp(-\frac{\|x_m - x_n^2\|^2}{\sigma^2}) = 1-\frac{\|x_m\|^2+\|x_n\|^2-2 < x_m, x_n >}{\sigma^2}+\mathcal{O}(\frac{1}{\sigma^4})$$

Now substituting this expansion into Eq.2.4 closes the proof, proving that the two update rules Eq.2.4 and Eq.2.5 (i.e. the two algorithms) are equivalent:

$$\begin{aligned}
u_i^{t+1} =& \arg\min_j \frac{1}{|w_j^t|^2}\sum_{m,n} K_{m,n}\delta(u_m^t, j)\delta(u_n^t, j) - \frac{2}{|w_j^t|}\sum_m K_{i,m}\delta(u_m^t, j) \\
=& \arg\min_j \frac{1}{|w_j^t|^2}\sum_{m,n} 1 - \frac{\|\underline{x}_m\|^2+\|\underline{x}_n\|^2-2\langle \underline{x}_m, \underline{x}_n\rangle}{\sigma^2}\delta(u_m^t, j)\delta(u_n^t, j) \\
& -\frac{2}{|w_j^t|}\sum_m 1 - \frac{\|\underline{x}_i\|^2+\|\underline{x}_m\|^2-2\langle \underline{x}_i, \underline{x}_m\rangle}{\sigma^2}\delta(u_m^t, j) \\
=& \arg\min_j 1 - \cancel{\frac{2}{|w_j^t|}\sum_m \frac{\|\underline{x}_m\|^2}{\sigma^2}} + \frac{2}{\sigma^2|w_j^t|^2}\sum_{m,n}\langle \underline{x}_m, \underline{x}_n\rangle\delta(u_m^t, j)\delta(u_n^t, j) \\
& -2 + \frac{2\|\underline{x}_i\|^2}{\sigma^2} + \cancel{\frac{2}{|w_j^t|}\sum_m \frac{\|\underline{x}_m\|^2}{\sigma^2}} - \frac{4}{|w_j^t|}\sum_m \langle \underline{x}_i, \underline{x}_m\rangle\delta(u_m^t, j) \\
=& \arg\min_j -1 + \frac{2\|\underline{x}_i\|^2}{\sigma^2} + \frac{2}{\sigma^2}\Bigg((\frac{1}{|w_j^t|^2}\sum_{m,n}\langle \underline{x}_m, \underline{x}_n\rangle\delta(u_m^t, j)\delta(u_n^t, j) \\
& - \frac{2}{|w_j^t|}\sum_m \langle \underline{x}_i, \underline{x}_m\rangle\delta(u_m^t, j)\Bigg) \\
=& \arg\min_j \frac{1}{|w_j|^2}\sum_{m,n}\langle \underline{x}_m, \underline{x}_n\rangle\delta(u_m, j)\delta(u_n, j) - \frac{2}{|w_j|}\sum_m \langle \underline{x}_i, \underline{x}_m\rangle\delta(u_m, j)
\end{aligned}$$

2.1.3 K-medoids Algorithm

K-medoids algorithm is a popular variation of k-means where each prototype is forced to be one sample i.e. we look for the N_C objects in the dataset that minimize the distance among them and their closest samples. Solving such partitioning problem, as in the case of k-means is an NP-hard problem and several heuristics were proposed in order to find sub-optimal solutions starting from an initial set of medoids.

For example, as originally proposed in [37] one may iteratively swap a random sample $\underline{x} \in \mathbf{X} \setminus \mathbf{W}$ with a random medoid $\underline{m} \in \mathbf{W}$ discarding all the swaps that do not decrease the quantization error. A more efficient faster heuristic that closely resembles the two steps EM-like procedure of k-means was also recently proposed [38]:

1. Update labels $u_i \leftarrow \arg\min_j ||\underline{x}_i - \underline{w}_j||^2$
2. Update medoids $\underline{w}_j \leftarrow \arg\min_{\underline{m}} \sum_{i=1}^{N} ||\underline{m} - \underline{x}_i||^2 \delta(u_i, j)$

One can iterate such algorithm until the stopping condition is reached i.e. $u_i^{t+1} = u_i^t$, $\forall i \in [1, N]$. It is worth observing that k-medoids does not require an explicit feature space for the data, indeed both the above steps can be carried out knowing the euclidean distance among samples or more generally a given distance matrix $D_{i,j} = D^2(\underline{x}_i, \underline{x}_j) \forall i, j \in [1, N]$.

2.1.4 The initialization problem

It is worth observing that all the mentioned algorithms are heuristics that find sub-optimal solutions starting from an intial set of prototypes $\mathbf{W}^0$. The quality of the solution therefore is closely related to the quality of the initialization technique used and a multi-start approach may be needed. A powerful initialization techniques is the one known as k-means++ [39] where an initial set of prototypes is selected with the following iterative procedure:

1. Pick a random sample $\underline{x} \sim P(\underline{x}) = \frac{1}{N}$ and add it to the prototypes set i.e. $\mathbf{W}^0 \leftarrow \mathbf{W}^0 \cup \{\underline{x}\}$
2. Compute the distance of each sample to its closest prototype: $D_m^2(\underline{x}) \leftarrow \min_{\underline{w} \in \mathbf{W}^0} D^2(\underline{x}, \underline{w})$
3. Pick a random sample from $\mathbf{X} \setminus \mathbf{W}^0$ i.e. $\underline{x} \sim P(\underline{x}) = \frac{D_m^2(\underline{x})}{\sum_j D_m^2(\underline{x}_j)}$ and add it to the prototypes set i.e. $\mathbf{W}^0 \leftarrow \mathbf{W}^0 \cup \{\underline{x}\}$
4. Iterate 2-3 until N_C samples have been selected as prototypes.

As shown by the authors, the above procedure dramatically reduce both the iterations needed by the algorithm to converge and the variance of the obtained results. Moreover selecting the set of initial prototypes as a subset of the samples can be effectively used for k-means, k-medoids and even in kernel space using as a distance measure the euclidean distance in the transformed space i.e. $D^2(\underline{x}, \underline{m}) = K(\underline{x}, \underline{x}) + K(\underline{m}, \underline{m}) - 2K(\underline{x}, \underline{m})$.

2.1.5 Relevant large scale techniques

In order to cope with the growing size of the data sets, several large scale techniques have been proposed in the literature. We are going to introduce here one rather successful approach, namely the one where the original dataset is divided into smaller mini-batches. Those mini-batches can be processed either sequentially reducing the memory footprint of the algorithm or in a distributed environment thus introducing significant speedups.

With this respect Sculley [40] proposed to use rather small mini-batches that are sequentially iterated as a series of Stochastic Gradient Descent (SGD) steps obtaining the following algorithm:

1. Load a random sampled mini-batch $\mathbf{M} \subset \mathbf{X}$
2. Initialize mini-batch labels $u_l = \arg\min_j ||\underline{x}_l - \underline{w}_j||^2 \ \forall \underline{x}_l \in \mathbf{M}$
3. For all $\underline{x}_l \in \mathbf{M}$:
    - Select proper cluster $j \leftarrow u_l$
    - Update count $|w_j| \leftarrow |w_j| + 1$
    - SGD step $\underline{w}_j \leftarrow (1 - \frac{1}{|w_j|})\underline{w}_j + \frac{1}{|w_j|}\underline{x}$
4. Go to 1.

The algorithm is usually iterated for a fixed number of iterations T and proved to work better than a simple online SGD approach. However the number of iterations T may be difficult to estimate a priori and the technique is intrinsically serial.

Another possibility is the one proposed in the series of works on Patch Clustering [41, 42] and Kernel Patch Clustering [43] in the context of Neural Gas (NG) algorithm which can be viewed as a weighted variant of k-means. There the dataset is divided into larger disjoint mini-batches that can be processed in parallel on a distributed system with N_P nodes, the procedure follows:

1. Divide the dataset in B disjoint mini-batches
2. Distribute next N_P mini-batches, one per node
3. Each node iterates the NG algorithm until convergence on its mini-batch
4. Gather all the $N_P N_C$ mini-batches centroids $\underline{w}_j^i$, $i \in [0, N_B]$, $j \in [0, N_C]$
5. Merge such mini-batches centroids into N_C global centroids $\underline{w}_j$.
6. Feed $\underline{w}_j$ to the next mini-batches as weighted samples with weights proportional to $\frac{|w_j|}{N_p}$.
7. If there are still unprocessed mini-batches go to 2.

Such kind of strategy has the advantage of being trivially parallel. In the method proposed by Sculley the merging phase was seamlessly carried out with the initialization step, here instead an explicit merging phase is needed since each node outputs an unrelated set of centroids.

2.2 Building Molecular Dynamics Coarse Grained Models

In the previous section we introduced the problem of clustering in the general settings of UL as the problem of finding a discrete representation of the data in the form of a small number of prototypes. It is therefore obvious how such problem is relevant in the context of MD simulations where one constantly faces the issue of defining few macro- or meso-states through which the system evolves starting from a large set of conformational frames. With this respect, in the following we are going to discuss how clustering analysis is indeed performed on MD trajectories in order to obtain coarse grained models of the underlying process.

2.2.1 MD data sets

As a starting point let us discuss the nature of a molecular dynamics trajectory from the standpoint of the clustering analysis. The first observation to be made regards the size and the dimensionality of the data, N being related with the length of the simulation and $d = 3N_a$ being related to the number of atoms N_a. With state of the art computational facilities one can expect N to be in the range $[10^7, 10^{10}]$ (i.e. microsecond long trajectories with fs time-step) with N_a ranging in between $[10^2, 10^5]$ depending on the application. Of course one has to keep in mind that those numbers are meant to increase as faster computational platforms will be available.

It is also worth stressing the fact that while the output of an MD simulation naturally lies on the $3N_a$-dimensional vector space spanned by the Cartesian coordinates of each atom, this may not be a convenient space to perform clustering. Indeed one is usually interested in conformational changes within the molecular system regardless any kind of rigid transformation. A standard distance metric in the field is the Minimum Root Mean Square Deviation (RMSD) defined as:

$$\mathrm{RMSD}(\underline{x}_i, \underline{x}_j) = \min_{\mathbf{RT}} \frac{1}{N_a} \sum_{l=1}^{N_a} ||\underline{x}_i^l - \underline{x}_j^l||^2 \tag{2.9}$$

where the minimum is taken on the set of all possible roto-translations. In this sense one may think at molecular conformations as structured graph-like data where a distance metric is properly defined but an explicit vector space is not readily available. If needed, an explicit feature space can be obtained by means of a featurization procedure:

1. Pick a reference frame

2. For each other frame:

- Find the best alignment with the reference.
- Compute explicit coordinates as displacement from reference.

This kind of procedure however is biased towards the alignment of the reference frame and in general does not guarantee a meaningful vector space where to perform the clustering analysis.

2.2.2 Clustering MD trajectories

From the above considerations one should understand how MD trajectories are large datasets in an intrinsic unknown high dimensional conformational space where a pairwise distance matrix can be obtained by means of the RMSD evaluation.

We already discussed how biomolecular processes of interest usually evolves through a series of metastable states defined as local minima of a given Free Energy Surface (FES). Being an MD trajectory obtained by sampling such FES, a certain smoothness is expected. Given a set **W** of prototypical conformations of the system describing those metastable states one may as well model the likelihood of the MD trajectory by means of a simple guassian model:

$$P(\mathbf{X}|\mathbf{W}) \propto e^{-\sum_{i=1}^{N}\sum_{j=1}^{N_C} \mathrm{RMSD}^2(\underline{w}_i,\underline{x}_j)} \tag{2.10}$$

Even though such assumption is quite naive, it does help us in selecting a proper clustering algorithm. In particular the k-means related techniques discussed in the previous section seems appropriate, solving a minimization problem in the form of:

$$\min_{\mathbf{W}} \sum_{i=1}^{N}\sum_{j=1}^{N_C} D^2(\underline{w}_i,\underline{x}_j)\delta(u_i,j) \tag{2.11}$$

This can be viewed as a maximization problem of the likelihood described by Eq.2.10 with the additional hard assignment of labels provided that one evaluates distances with the proper RMSD metric. This last requirement rules out the possibility of using k-means in its standard formulation thus identifying kernel k-means and k-medoids as two valuable methods.

K-medoids is indeed one of the tools of choice when one looks into the literature of MD clustering. In particular we highlight in this context the achievement of Decherchi et al. in [1], where microsecond-long trajectories of the binding mechanism of a drug, specifically a transition state analogue named DADMe-immucillin-H, to the Purine Nucleoside Phosphorylase (PNP) enzyme were analyzed. Clustering was there performed with an in-house version of the k-medoids algorithm presented before and the binding mechanism was elucidated running a shortest path analysis on the graph of connected clusters where the edge weights were set to the negative logarithm of the number of observed transitions.

2.2.3 Markov State Models

Performing plain clustering on MD is therefore a valuable solution to obtain a coarse grained model of the simulated process. Starting from this consideration, one may think to use clustering analysis in order to build a more sophisticated kinetic model where a network of conformational states is related to a probability matrix describing transitions between them. This is the aim of a particularly successful class of models known as Markov State Models (MSMs) [4, 44] .

The main idea behind MSMs is to first construct a reduced dynamics description by clustering MD trajectories into a large set of microstates and then to further coarse grain such description in a kinetically meaningful way in order to build a more understandable macrostates model. The overall procedure goes as follow:

1. Perform a clustering step e.g. via k-medoids algorithm with RMSD metric in order to cluster simulation data into microstates (i.e. small portion of the conformational space with non vanishing probability).

2. Convert the MD trajectory into a time series of microstates labeling each frame within the simulation with the proper microstate.

3. Build a count matrix $\mathbf{C}(\tau)$ whose elements $C_{ij}(\tau)$ represent the number of transitions observed between state i and state j within a lag time τ.

4. Infer a transition matrix $\mathbf{T}(\tau)$ by means of maximum likelyhood analysis on the count matrix $\mathbf{C}(\tau)$

5. Coarse grain the model with a second clustering step in order to obtain macrostates which give clear insights about the kinetic of the process.

It should be clear that the accuracy of MSMs highly depends on the quality of the initial data set (i.e. how well transitions between microstates are sampled). However it is worth noting how MSMs can be used to guide further data acquisition improving the overall sampling. Indeed the count matrix $\mathbf{C}(\tau)$ can be used to predict the statistical error on $\mathbf{T}(\tau)$ and, consequently, the states that are limiting the accuracy of the model.

2.3 Large Scale Kernel K-means for MD

Above we discussed how clustering represents a valuable technique for MD trajectory analysis both as a standalone modeling tool and as core procedure in the more sophisticated MSMs. We also discussed how the requirements of MD data would suggest kernel k-means as a possible technique of interest since firstly it does not require an explicit feature space and secondly, it recovers standard k-means results as limiting case.

Computational complexity and memory occupancy however are major drawbacks of kernel based clustering where the size of the kernel matrix to be stored together with the number of kernel function evaluations scales quadratically with the number of samples. Such computational burden has historically limited the success of kernel k-means as an effective clustering technique. From this considerations we naturally identify the first out of two main objectives of the thesis.

Purpose of the thesis will be the development of a new clustering engine to perform large scale kernel k-means on High Performance Computing (HPC) facilities. The developed tool will be a valuable asset not only in MD data analysis but more generally in the context of clustering where the theoretical capabilities of kernel k-means have been already demonstrated [45].

Chapter 3

Molecular Dynamics Inspiring Unsupervised Learning

In the previous chapter we presented a first connection between Molecular Dynamics (MD) and Unsupervised Learning (UL) showing how Clustering techniques can be effectively used in order to learn coarse grained models that facilitate the interpretation of MD simulations. With this respect, we defined one of the two objective of the thesis i.e. developing an efficient large scale clustering engine that meets the requirements of MD data.

In this chapter we will discuss a further, more profound connection between the two fields of interest showing how not only MD resides within the applicability domain of UL but how it is also able to inspire totally new learning problems. As a first step in this direction we will introduce the concepts of Minimum Energy Path (MEP) and Minimum Free Energy Path (MFEP) proper of statistical mechanics. In doing so we will state why they are relevant in the domain of MD simulations and we will discuss the algorithms available in the literature for computing them. As a second step we will then give an overview of a closely related problem in the field of UL, namely the problem of finding Principal Curves (PCs).

As a last point we will discuss how the transposition of the MFEP concept into the domain of UL brings it close to the one of PC. Such discussion will lead to the definition of a new cognitively sound concept i.e. the one of Principal Path in data space. The theoretical derivation of an algorithm to find such Principal Path is identified as the second and final objective of the thesis thus closing this introductory part.

3.1 Minimum Energy Path

Let us consider a molecular system evolving according to a given potential energy $U(\underline{x})$ where the state $\underline{x}$ is a $3N_a$ dimensional vector describing the cartesian coordinates of all the atoms in the system. The MEP is defined as the steepest path connecting two local minima of $U(\underline{x})$ i.e. $\underline{x}_A$ and $\underline{x}_B$, via a saddle point. It follows by the definition of steepest path that the force $\underline{F} = -\underline{\nabla}U(\underline{x})$ has to be tangent to the MEP everywhere. Representing the path with a parametric curve $\underline{x}(\alpha)$,$\alpha \in [0, 1]$ one can easily write the following differential equation:

$$(\nabla U(\underline{x}(\alpha)))_\perp = 0 \; \forall \alpha \in [0, 1] \tag{3.1}$$

which can be solved in order to find the MEP simply adding the two boundary conditions: $\underline{x}(0) = \underline{x}_A$ and $\underline{x}(1) = \underline{x}_B$.

MEPs are relevant objects in the study of a simulated process of interest where they are used in the definition of *reaction coordinates* in order to quantitatively describe the transitions $\underline{x}_A \leftrightarrow \underline{x}_B$. It is obvious to observe that along the MEP the maximum value for the potential $U(\underline{x})$ is reached at the saddle point $\underline{x}_s$ which is usually identified with an intermediate transition state of the process. The energy differences $U(\underline{x}_s) - U(\underline{x}_A)$ and $U(\underline{x}_s) - U(\underline{x}_B)$ describe the activation energy barriers of the reaction and are of paramount importance for example in the estimate of transition rates.

From all the above considerations it stems the need for computational tools in order to evaluate the MEP along a molecular simulation. Among several possible techniques in the following we will limit ourselves to give an overview on *chain of states* methods since they will be the ones relevant for latter discussions. Those kinds of techniques revolve around the idea of having R replicas of the system $\underline{x}_1, \dots .\underline{x}_R$ connected to form a 1D topology. Such chain of replicas is usually initialized with a simple guess for the MEP e.g. straight line connecting $\underline{x}_A$ to $\underline{x}_B$ and it is evolved according to a dynamics that slowly converges towards a discretized MEP as shown in Fig.3.1.

3.1.1 The plain elastic band method

The simplest *chain of states* method is the one known as Plain Elastic Band (PEB) [46] where the 1D topology among replicas is enforced with a set of harmonic restraints. In such scenario the global potential energy of the system is:

$$\hat{U}(\underline{x}_1, \dots, \underline{x}_R) = \sum_{i=0}^{R} U(\underline{x}_i) + \frac{k}{2} \sum_{i=0}^{R-1} (\underline{x}_{i+1} - \underline{x}_i)^2 \tag{3.2}$$

One can now numerically integrate the equation of motions i.e. performing an MD simulation for the connected R replicas evolving according

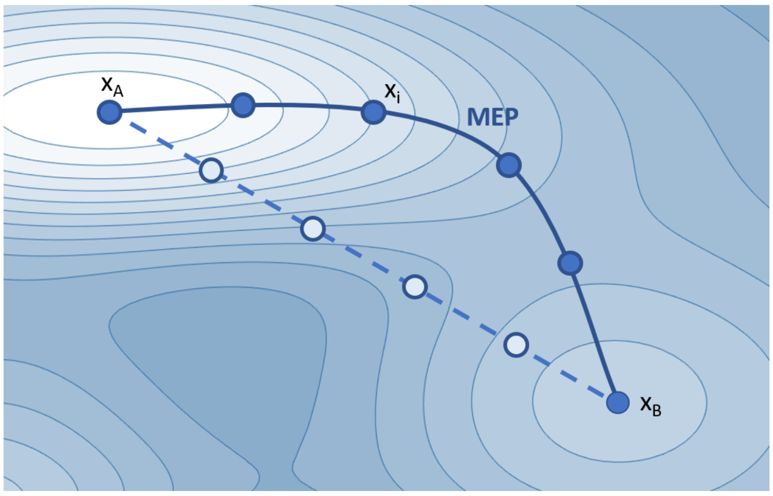

Figure 3.1: Pictorial representation of chain of states methods for finding the MEP. The replicas are initialized on a straigth path connecting x_a and x_b and evolve towards a piece-wise approximation of the MEP.

to such global potential in order to obtain a discretized approximation to the MEP. The approach works provided that one is able to select a proper value for the spring constant k. The choice of an optimal value for such parameter is not trivial at all as shown in [47], indeed while large values of k are desirable in order to enforce a smooth curve, small values of k are also desirable in order to do not steer away from the actual MEP.

3.1.2 The nudged elastic band method

In order to solve the problems deriving from the choice of a proper value for the spring constant k the Nudged Elastic Band (NEB) [47, 48] method was introduced. The idea behind NEB is quite simple, one starts observing that the force acting on each replica of a Plain Elastic Band model can be decomposed in the following 4 contributions:

$$F^{\text{PEB}} = F_{\perp} + F_{||} + F_{\perp}^{k} + F_{||}^{k} \tag{3.3}$$

where the force F derived from the original potential and the force F^{K} derived by the harmonic restraints are decomposed along the directions tangent and perpendicular to the path. At this point one integrates out the contribution of $F_{||}$ and $F_{\perp}^{k}$ thus applying the following *nudged* force on each replica of the system:

$$F^{\text{NEB}} = F_{\perp} + F_{||}^{k} \tag{3.4}$$

As extensively discussed in [47] such modified dynamics enforces the equal spacing of replicas via $F_{||}^{K}$. However since such force is acting by construction in the direction parallel to the path the replicas are not steered away from the steepest descent dynamics described by $F_\perp$. It is worth observing that even though the technique heuristically works as widely demonstrated in the literature, the physical interpretation of such *nudged* dynamics is not clear.

3.2 Minimum Free Energy Path

As discussed in chapter 1, one is usually interested in simulating a system at a given temperature, thus closely reproducing the conditions observed in actual experiments. In such setup one is studying the transitions among *thermodinamical* states defined as average on finite portion of the available phase space. Entropy is therefore a relevant factor and the interest is shifted to Free Energy Differences and Free Energy Paths in order to characterize the process. Both the above techniques deal with the system in the space of Cartesian coordinates and therefore seems inappropriate for the task. Indeed as discussed in chapter 1 the Free Energy Surface (FES) is usually reconstructed on a set of reduced descriptors i.e. a set of Collective Variables (CVs) $\underline{z} = (\theta_1(x), \dots, \theta_d(x))$. In such space the MFEP is defined as the steepest trajectory connecting two local minimum of the Free Energy $\underline{z}_A$ and $\underline{z}_B$ via a saddle point.

It is possible to demonstrate that in the CVs space the constitutive equation 3.1 can be written as:

$$(\mathbf{M}(\underline{z}(t))\nabla_z \mathcal{F}(\underline{z}(t)))_\perp = 0 \tag{3.5}$$

with boundary conditions $\underline{z}(0) = \underline{z}_A$, $\underline{z}(1) = \underline{z}_B$ where $\mathbf{M}(\underline{z})$ is the average jacobian of the transformation $\underline{z} = (\theta_1(x), \dots, \theta_d(x))$. We will now briefly discuss a chain of states method to evaluate the MFEP common in the field of MD, namely the String Method.

3.2.1 The string method

The string method as introduced in [49] looks for the MFEP starting from an initial guess and then evolves a simple steepest descent dynamics derived by Eq.3.5 on each replica $\underline{z}_i$ of the system i.e.

$$\underline{z}_i \leftarrow \underline{z}_i - \Delta t(\mathbf{M}(\underline{z}_i)\nabla_z \mathcal{F}(\underline{z}_i))_\perp \tag{3.6}$$

It should be clear that at each iteration of such procedure one needs to evaluate the the mean force $-\nabla_z \mathcal{F}(\underline{z}_i)$ and the jacobian $\mathbf{M}(\underline{z}_i)$ for every replica. This can be done for example with a set of restrained MD simulations where the total potential energy of each replica is given by $\hat{U}(\underline{x}_i) = U(\underline{x}_i) + k(\underline{\theta}(\underline{x}_i) - \underline{z}_i)^2$.

The authors of the method also suggest to introduce a smoothing step in order to prevent abrupt fluctuations of the path i.e.

$$\underline{z}_i \leftarrow (1-s)\underline{z}_i + \frac{s}{2}(\underline{z}_{i-1} + \underline{z}_{i+1}) \tag{3.7}$$

with $s \in [0,1]$ being a smoothing parameter to be set.

At last one can introduce a further reparametrization step in order to enforce an equally spaced sampled path. The overall 4 steps procedure can be summarized as:

1. Estimate $-\nabla_z \mathcal{F}(\underline{z}_i)$ and $\mathbf{M}(\underline{z}_i)$ $\forall i$.
2. Evolve each replica according to Eq.3.6.
3. Prevent abrupt fluctuations applying Eq.3.7.
4. Reparametrize the curve enforcing equal arc-length among subsequent replicas.

More sophisticated approaches were then developed starting from this idea in order to improve the sampling process needed for the estimates of $-\nabla_z \mathcal{F}$ and $\mathbf{M}$ (see e.g [50]) however for the sake of our future discussions the simple formulation above is sufficient. We close this section recalling that in the same work the authors also prove that the MFEP coincides with the maximum likelihood reaction path thus highlighting its importance in the context of theoretical and computational chemistry.

3.3 Principal Curves

Let us now leave aside the problem of finding MEP and MFEP in the context of molecular simulations to focus instead on a related problem in the literature of UL, namely the problem of finding PCs. PCs were intuitively defined by Hastie [51, 52] as smooth one-dimensional curves that pass through the *middle* of the data. A mathematically formal definition was also given by the same author introducing the concept of self consistency that we are going to review now.

Let us consider a set of sampled data $\underline{x}_i \in \mathbb{R}^d$, and a parametric curve $\underline{f}(\alpha)$ lying in the same space. We define the projection index $\alpha_f(\underline{x}_i)$ as the value of α for which $\underline{f}(\alpha)$ is closer to $\underline{x}_i$ i.e.

$$\alpha_f(\underline{x}_i) = \inf_{\alpha} ||\underline{x}_i - \underline{f}(\alpha)|| \tag{3.8}$$

Then a curve $\underline{f}(t)$ is called self-consistent or PC if $E(\underline{x}|\alpha_f(\underline{x}) = \alpha) = \underline{f}(\alpha)\forall\alpha$ i.e. if the expected value of the data projection onto the curve coincide with the curve itself. Such definition naturally leads to the following Hastie-Stuetzle two step procedure for finding PCs starting from an initial guess $\underline{f}^0(\alpha)$:

1. Compute projection indices $\alpha_{f^t}(\underline{x}) = \inf_\alpha ||\underline{x}_i - \underline{f}^t(\alpha)||$
2. Update the curve $\underline{f}^{t+1}(\alpha) = E(\underline{x}|\alpha_{f^t}(\underline{x} = \alpha))$

where one continuously updates each point of the curve with the expected value of the samples projecting onto it until a stopping condition is met. Such algorithm however may work just in the hypothetical case of an infinite sample size and cannot be applied in practice as it is. Indeed in a finite sample case one can expect at most one data point to project on a given α of the curve resulting in the impossibility of performing the expectation step. Moreover some sort of discretization of the curve is needed in order to deal with a finite number of parameters. Several successful algorithms have been proposed in order to deal with both those problems starting from the original idea of Hastie and Stuetzle, see among others [53, 54, 55]. In the following we will focus our attention on two popular techniques where a set of prototypes $\mathbf{W} = \{\underline{w}_i\}$, connected to form a one-dimensional topology, is optimized to find the PC.

3.3.1 Elastic maps

Elastic maps [56, 57, 58] were developed as systems of elastic graphs optimized in data space to find low dimensional data embeddings. The output of such methods is in the form of regular grids in data space that can effectively approximate non-linear principal manifolds in the Hastie and Stueltze sense.

Let G be an undirected graph with N_C vertices v_i and a set of edges E. We now introduce the map $\phi : V \to \mathbb{R}^d$ as a function which embeds the graph vertices into the data space $\mathbb{R}^d$. The approximation energy of the map is defined as:

$$U_A(G, \mathbf{X}) = \sum_{i=1}^{N} \sum_{j=1}^{N_C} ||\underline{x}_i - \underline{\phi}(v_j)||^2 \delta(u_i, j) \tag{3.9}$$

By analogy with the k-means cost function introduced in the previous chapter we will refer in the following to $\underline{w}_j = \underline{\phi}(v_j)$ as to the *prototypes* of the map. Let us now introduce the elastic energy and the bending energy of the map defined as:

$$\begin{aligned} U_E(G) &= \sum_{e_i \in E} \lambda_i ||\phi(e_i(0)) - \phi(e_i(1))||^2, \\ U_B(G) &= \sum_{s_j \in S^k} \mu_{jk} ||\sum_{i=1}^{k} \phi(s_j(i)) - k\phi(s_j(0))||^2 \end{aligned} \tag{3.10}$$

where S^k is a family of k-star sub-graphs of G.

In their most general form elastic maps deal with the minimization of the following global energy:

$$U(G) = U_A(G, \mathbf{X}) + U_E(G) + U_B(G) \tag{3.11}$$

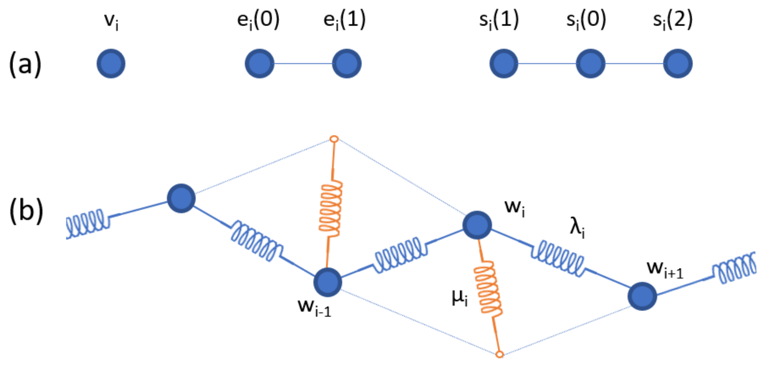

Figure 3.2: (a)Schematic representation of the constitutive elements of a one-dimensional elastic map i.e. vertices v_i, edges e_i and 2-star subgraphs s_i. (b) Schematic representation of a one-dimensional elastic map for PC learning.

For the purpose of PC Learning elastic maps are usually used considering a one dimensional chain with only 2-star contributions as shown in Fig.3.2 . Therefore the optimization problem to be solved has the following form:

$$\min_{\mathbf{W}} \sum_{i=1}^{N} \sum_{j=1}^{N_C} ||\underline{x}_i - \underline{w}_j||^2 \delta(u_i, j) + \\ + \sum_{i=1}^{N_C-1} \lambda_i ||\underline{w}_{i+1} - \underline{w}_i||^2 + \sum_{i=2}^{N_C-1} \mu_i ||\underline{w}_{i-1} + \underline{w}_{i+1} - 2\underline{w}_i||^2 \tag{3.12}$$

The authors propose an Expectation Maximization (EM) procedure similar to the one used in standard k-means clustering where 1) the labels u_i^t are computed for a given set of prototypes and 2) the updated prototypes $\mathbf{W}^{t+1}$ are obtained minimizing Eq.3.12 given u_i^t. Such two steps are iterated until a given stopping condition is reached e.g. the change in the cost function becomes less than a small value ϵ.

One can immediately recognize how Eq.3.12 is a regularized form of the standard k-means minimization problem where the set of $2N_C - 3$ regularizing parameters $\{\lambda_i, \mu_j\} \forall i \in [1, N_C - 1], \forall j \in [2, NC - 1]$ is introduced. The major drawback of elastic maps has to be found precisely in the large number of such parameters resulting in a too wide solution space and in the lack of a proper model selection framework to set them.

3.3.2 Self Organizing Maps

Probably one of the most widely used frameworks to learn principal manifolds is the one of Self Organizing Maps (SOMs) that we are going to introduce here.

The method was originally proposed by Kohonen in [59] as a competitive learning procedure to train a neural network where each neuron has a weight vector $\underline{w}_i \in \mathbb{R}^d$ and is connected to the other neurons according to a given neighborhood set $\mathcal{N}_i$.

The stochastic training process goes as follows:

1. Select a random data sample $\underline{x}_i$
2. Find the best matching neuron $u_i = \arg\min_j ||\underline{x}_i - \underline{w}_j||^2$
3. Update the best matching neuron and its neighborhood set according to:
$$\underline{w}_j^{t+1} = (1-\lambda)\underline{w}_j^t + \lambda \underline{x}_i \ , \lambda \in [0,1] \ , j \in \mathcal{N}_{u_i} \tag{3.13}$$

If one sets the neighborhood set to be equal to $\mathcal{N}_i = \{\underline{w}_{i-1}, \underline{w}_i, \underline{w}_{i+1}\}$ it is clear how such procedure describe the evolution of a one-dimensional chain of prototypes in data space, where the prototypes are the actual weight vectors of the neurons. As pointed out by Kegl in [60], even though the method was developed in the context of competitive neural networks, one-dimensional SOMs can be effectively used in order to approximate PCs in the Hastie-Stuetzle sense. However, it is worth stressing the fact that a major short coming of SOMs is the absence of an objective function to be minimized. Indeed the learning procedure it is completely heuristic and a functional formulation cannot be found [61].

3.4 From MFEPs to Principal Paths in Data Space

We will now discuss how the concepts introduced in this chapter can be jointly used to intuitively define the notion of Principal Paths in data space. Let us start observing that MEP techniques relies on the knowledge of the underlying potential energy of the system $U(\underline{x})$ in order to simulate a set of replicas along the path. Similarly, algorithms for finding the MFEP relies on the ability of sampling the equilibrium distribution $e^{-\beta\mathcal{F}(\underline{z})}$ around the path by means of MD simulations. In both cases such techniques are on-line procedures where the path is optimized together with a set of replicas of the system.

Let us now change perspective, assuming that a set of samples was generated beforehand for example by means of a long MD simulation. How can we now define energetically relevant paths in such data space?

1. In analogy to the MEP and the MFEP, an energetically relevant path in data space have to be defined in relation to fixed boundary conditions i.e. fixed starting and ending samples $\underline{x}_A$ and $\underline{x}_B$.

2. Since the MFEP is the maximum likelyhood reaction path, it will also be the most probable transition path $\underline{z}_A \rightarrow \underline{z}_B$ to be sampled along a dynamic simulation. We can speculate that, no matter how rare such transition is, if it is observed (and the CVs are proper smooth descriptors of the system) then the samples produced will populate contiguous regions of the data space. An energetically relevant path in data space therefore have to pass through those populated regions.

Starting with this two observations, we can intuitively define the notion of Principal Path in data space as a smooth path connecting a starting sample $\underline{x}_A$ to an ending one $\underline{x}_B$ locally passing through the *middle* of the data. A Principal Path can be thought of as a local version of the PC where local means that a valuable path may be found that passes through just a subset of the data.

Purpose of the thesis will be the formalization of such intuitive definition introducing both a functional that embeds the notion of Principal Path and an effective optimization algorithm. The developed technique will be a valuable tool both in MD where it can be used to infer coarse grained models out of long trajectories and, more generally, in the context of manifold and topological methods, a branch of UL with growing interest [20].

Part II

Developed Methods

Chapter 4

Distributed Kernel K-means

The first three chapters of the thesis served to shape a challenging, multidisciplinary research field, namely the one formed by the cross contamination of Molecular Dynamics (MD) and Unsupervised Learning (UL). Within the realm of computational chemistry we showed in chapter 1 how MD is an effective *in-silico* tool to generate large collections of molecular conformations. In the same chapter we then explained how the urge to organize such data into coarse grained readable models brought MD into the applicability domain of UL techniques, in particular of clustering. Among several possible algorithms we discussed how kernel k-means represents a favorable choice in this context, without requiring an explicit vector space for the data and having a simple definition of clusters (that matches the nature of MD trajectories).

We enter, with the following chapter, the details of our original contributions to the field. Hereafter we introduce a novel clustering engine, namely Distributed Kernel K-means (DKK), to perform large scale kernel k-means clustering. A two-fold approximation technique is presented to tackle the well known $O(N^2)$ scaling of exact kernel methods. We then show how such approximation technique, can be effectively parallelized entering in the details of an ad hoc distribution strategy. As it will be clear, the twofold approximation introduced is controlled via two straightforward parameters: the number of mini-batches B and the sparsity degree of the centroid representation s. These two knobs allow the user to finely adapt the algorithm to the available computational resources in order to cope with virtually any sample size.

Throughout the chapter the following notation will be used:

- N is the number of samples.
- N_C is the number of cluster prototypes.
- $\phi(\cdot) : \mathbb{R}^d \to \mathbb{R}^{d'}$ is the possibly non-linear transformation mapping the d-dimensional input space into a d'-dimensional transformed one.

- $\mathbf{X}$ is the $N \times d$ matrix of samples x_i arranged in a row wise fashion i.e. $\mathbf{X}_{i,\cdot} = \underline{x_i}$.
- $\mathbf{K}$ is the $N \times N$ kernel matrix defined as $\mathbf{K}_{i,j} = \langle \phi(\underline{x}_i), \phi(\underline{x}_j) \rangle$.
- $\mathbf{W}$ is the $N_C \times d'$ matrix of cluster prototypes $\underline{w}_i$ arranged in a row wise fashion.
- $|w_i|$ represents the cardinality of the i-th cluster.
- $u_i \in [1, N_C]$ is the label associated with the i-th sample $\underline{x}_i$.

4.1 The base algorithm: kernel k-means by Zhang and Rudnicky

As a starting point we briefly present here a reformulation of the Expectation Maximization (EM) procedure to minimize the kernel k-means cost function in terms of the cluster compactness $\underline{g}$ $(1 \times N_C)$ and the average cluster similarity $\mathbf{F}$ $(N \times N_C)$.

For the sake of clarity let us recall the kernel k-means cost function introduced in chapter 2:

$$\Omega(\mathbf{X}, \mathbf{W}) = \sum_{i=1}^{N} \sum_{j=1}^{N_C} \|\phi(\underline{x}_i) - \underline{w}_j\|^2 \delta(u_i, j)$$

For a given cluster j we define the compactness as:

$$g_j = \frac{1}{|\underline{w}_j|^2} \sum_{l,m} K_{l,m} \delta(u_l, j) \delta(u_m, j) \tag{4.1}$$

whereas the average similarity of a sample i with a cluster j is given by:

$$F_{i,j} = \frac{1}{|w_j|} \sum_{l} K_{i,l} \delta(u_l, j) \tag{4.2}$$

With these definitions, the EM minimization procedure for kernel k-means can be written as:

$$\begin{cases} u_{i,t+1} \leftarrow \arg\min_j g_{j,t} - 2F_{(i,j),t} \\ g_{j,t+1} \leftarrow \frac{1}{|\underline{w}_j|^2} \sum_{l,m} K_{l,m} \delta(u_{l,t+1}, j) \delta(u_{m,t+1}, j) \\ F_{(i,j),t+1} \leftarrow \frac{1}{|w_j|} \sum_l K_{i,l} \delta(u_{l,t+1}, j) \end{cases} \tag{4.3}$$

and the medoid approximation for a given cluster prototype j can be rewritten as:

$$\phi^{-1}(\underline{w}_j) \approx \underline{m}_j = \arg\min_{\underline{x}_l \in X} K_{l,l} - 2F_{i,j} \tag{4.4}$$

Such reformulation of the kernel k-means algorithm was originally proposed by Zhang and Rudnicky [62] to reduce the memory footprint of the kernel matrix allowing disk caching. As we are going to show, we took advantage of the same formalism to design an efficient distribution strategy.

4.2 A new two-fold approximation to kernel k-means

We introduce in this section a novel two-fold approximation for the kernel k-means minimization algorithm. First, we introduce a mini-batch approach that reduces the computational cost of a factor B with $\frac{N}{B}$ being the mini-batch size and B the actual number of mini-batches. An a priori sparse representation for the cluster centroids is then discussed allowing for a further reduction in the computational cost of a factor $\frac{1}{s}$ with $s < 1$ being related to the sparsity of the representation. The action of such two-fold approximation on the number of kernel elements to be evaluated is illustrated in Fig.4.1(c). One should immediately appreciate also how, reducing the number of such element also the memory footprint of the algorithm will be dramatically reduced.

Remark about the notation used: in the following, a superscript identifies a specific mini-batch quantity, when no superscript is used the quantity has to be intended as a global quantity. As an example, $\underline{w}_j^i$ represents the j-th cluster prototype for the i-th mini-batch whereas $\underline{w}_j$ is the j-th global cluster prototype obtained combining the partial results of all mini-batches.

4.2.1 Mini-batch approximation

Our primary approach to reduce the $O(N^2)$ complexity coming from the kernel matrix evaluation consists of splitting the dataset into disjoint mini-batches that are processed one after the other. The procedure can be summarized by the following steps that will be carefully explained in the subsequent paragraphs:

1. Fetch one mini-batch at a time (until all data is consumed).
2. Evaluate the mini-batch kernel matrix and initialize the mini-batch labels.
3. Iterate kernel k-means EM-procedure on one minibatch and collect results.
4. Merge together current minibatch results to global results with a proper strategy and go to step 1.

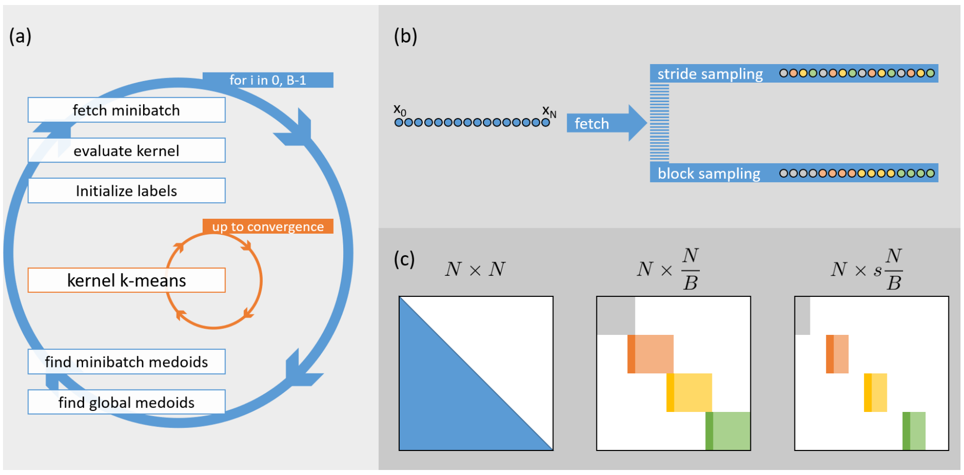

Figure 4.1: (a) Pictorial description of the algorithm to highlight its double loop structure. The iterations of the outer loop are fixed once B is set whereas the inner loop runs up to convergence. (b) Visualization of two possible sampling strategies to divide the dataset into mini-batches. (c) From left to right we visualize the effect of the two-fold approximation proposed on the number of kernel matrix elements that need to be evaluated. With standard kernel k-means the symmetry of the matrix can be exploited to evaluate just $\frac{N^2}{2}$ elements, introducing the mini-batch approximation one needs to evaluate $N \times \frac{N}{B}$ elements, introducing also the a priori sparse representation of cluster centers the number of kernel evaluations is cut to $N \times s\frac{N}{B}$.

Fig.4.1(a) shows a pictorial description of such algorithm highlighting its hierarchical structure. A pseudo code for the entire procedure is also provided in Alg.1 at the end of this section.

Mini-batch fetching The first sensible choice to be made, regards the way in which the dataset is divided in B disjoint mini-batches of size $\frac{N}{B}$. A variety of possibilities arise, we present here two common reasonable sampling strategies.

Let us assume that the dataset $\mathbf{X} = \{\underline{x}_1, ..., \underline{x}_N\}$ is generated by a discrete time process $\mathbf{X}_t$ sampled at time $t = \{\Delta t, ..., N\Delta t\}$. For example this is exactly the case of conformational frames obtained in a standard MD simulation. In such situation one can expect the autocorrelation of the process to decay after a lag time $\tau > \Delta t$ suggesting a stride sampling strategy of this kind: $\mathbf{X}^i = \{\underline{x}_{i+jB}\}, j \in [0, \frac{N}{B} - 1]$. This first approach suggests to split the data using a striding strategy to assure the best data distribution strategy among the mini-batches. This however requires waiting for the end of the simulation to start the clustering process.

A second way to approach the sampling is to feed the algorithm with a data stream; in such situation a simpler block sampling strategy is desirable in order to begin the clustering procedure as soon as the first $\frac{N}{B}$ samples are received i.e. $\mathbf{X}^i = \{\underline{x}_{i\frac{N}{B}+j}\}, j \in [0, \frac{N}{B} - 1]$. Obviously this second strategy has the drawback that each mini-batch is time-consistent and, as such, the sampling on each mini-batch is rather partial.

We will discuss within the experimental section the effects of those two choices in a worst case scenario (a concept drift case).

Kernel evaluation and mini-batch initialization Once a mini-batch is fetched, it is straightforward to evaluate the mini-batch kernel matrix $\mathbf{K}^i$ with a computational cost of $O(\frac{N^2}{B^2})$. Let us now discuss how it is possible to initialize the i-th mini-batch labels. We distinguish two cases:

$i = 0$ **:** during the first mini-batch the global cluster medoids have to be selected randomly or by means of some rational. We propose here to use the kernelized version of the popular k-means++ initialization scheme as it was introduced in chapter 2

$i \neq 0$**:** Starting from the second mini-batch the global cluster medoids $\mathbf{M} = \{\underline{m}_j \approx \phi^{-1}(\underline{w}_j)\}$ obtained at the end of the previous iterations are used for the initialization. Simply assigning each new sample to its closest global medoid we obtain:

$$u_l^i = \arg\min_j ||\phi(\underline{x}_l^i) - \underline{m}_j||^2 \tag{4.5}$$

$$= \arg\min_j [K^i(\underline{x}_l^i, \underline{x}_l^i) + K^i(\underline{m}_j, \underline{m}_j) - 2K(\underline{x}_l^i, \underline{m}_j)] \tag{4.6}$$

Such initialization step automatically allows to keep track of the clusters across different mini-batches. Indeed the global j-th medoid obtained at the end of the $(i-1)$-th iteration is used as initialization for the same j-th cluster of the i-th mini-batch. This avoids ambiguity also when the partial mini-batch result has to be merged with the global one: the mini-batch medoid $\underline{m}_j^i$ will be combined with the global medoid m_j having the same index j.

It should be understood that in order to evaluate the second term of such equation one has to perform additional computations. Indeed one has to compute the kernel function for all the pairs $(\underline{x}_l^i, \underline{m}_j)$ where $\underline{x}_l^i$ belongs to the i-th mini-batch and m_j is a global medoid coming from the previous $(i-1)$ mini-batches. It is therefore clear that the initialization phase of each mini-batch requires an auxiliary kernel matrix $\tilde{\mathbf{K}}^i$ of size $\frac{N}{B} \times N_C$ computed evaluating $K(\underline{x}, \underline{m})\ \forall \underline{x} \in \mathbf{X}^i, \underline{m} \in \mathbf{M}$.

Mini-batch inner EM loop Given a mini-batch kernel matrix $\mathbf{K}^i$ and an initial set of labels $\underline{u}_0^i$, equations 4.1 - 4.3 are used to perform a Gradient Descent (GD) optimization of the reduced cost function:

$$\Omega(\mathbf{X}, \mathbf{W}^i) = \sum_{\underline{x}_j \in \mathbf{X}^i} \sum_{l=1}^{N_C} \|\phi(\underline{x}_j) - \underline{w}_l^i\|^2 \delta(u_j^i, l) \tag{4.7}$$

A final set of labels $\underline{u}^i$ is obtained as a result of such optimization procedure. It is worth stressing the fact that at this point the set of mini-batch cluster prototypes is not explicitly known. As a matter of fact, even though we could formally write the equation: $\underline{w}_j^i = \frac{1}{|\underline{w}_j^i|} \sum_{\underline{x}_l \in \mathbf{X}^i} \phi(\underline{x}_l)\delta(u_l, j), j \in [1, N_C]$, without knowing the explicit form of ϕ, we would not be able to evaluate it. As a solution, we propose a medoid approximation as introduced in chapter 2 for standard kernel k-means. Therefore we set the cluster prototypes to be equal to:

$$\underline{w}_j^i \leftarrow \phi(\underline{m}_j^i) : \ \underline{m}_j^i = \arg \min_{\underline{x}_l \in \mathbf{X}^i} \|\phi(\underline{x}_l) - \underline{w}_j^i\|^2 \tag{4.8}$$

More sophisticated approaches based, for instance, on a sparse representation of cluster centers are possible (e.g. see [40]). However the inherent additional computational cost and the satisfactory results already obtained by means of the simple medoid approximation discouraged us to further investigate this possibility.

Full batch cluster centers update We discuss now on how to merge the medoids $\mathbf{M}^i$ of the i-th mini-batch together with the global medoid set $\mathbf{M}$.

Let $\{\underline{w}_j = \phi(\underline{m}_j)\}$ be the global medoids obtained from the $(i-1)$ previous iterations of the outer loop and let $\{\underline{w}_j^i = \phi(\underline{m}_j^i)\}$ be the approximated cluster centers for the current $i-th$ mini-batch. We propose to obtain the resulting global cluster prototypes as a convex combination of the two:

$$\underline{w}_j \leftarrow (1-\alpha)\phi(\underline{m}_j) + \alpha\phi(\underline{m}_j^i) \tag{4.9}$$

Practically, since Eq.4.9 cannot be evaluated directly, we introduce a second medoid approximation as already done in the previous paragraph so that:

$$\underline{w}_j \leftarrow \phi(\underline{m}_j) : \ \underline{m}_j = \arg \min_{\underline{x}_l \in \mathbf{X}^i} \|\phi(\underline{x}_l) - (1-\alpha)\phi(\underline{m}_j) - \alpha\phi(\underline{m}_j^i)\|^2 \tag{4.10}$$

The choice of this convex combination stems from a simple but important observation; indeed in order to choose the coefficient α let us consider the updating equation for the global cluster center $\underline{w}_j$ at the second iteration of the algorithm, when the first two mini-batches are merged in a single one (assuming this is the complete dataset):

$$\begin{aligned}
\underline{w}_j &= \frac{1}{|\underline{w}_j^0|+|\underline{w}_j^1|} \sum_{\underline{x}_i \in \mathbf{X}^0 \cup \mathbf{X}^1} \phi(\underline{x}_i)\delta(u_i, j) \\
&= \frac{|\underline{w}_j^0|}{|\underline{w}_j^0|+|\underline{w}_j^1|} \frac{1}{|\underline{w}_j^0|} \sum_{\underline{x}_i \in \mathbf{X}^0} \phi(\underline{x}_i)\delta(u_i, j) + \frac{|\underline{w}_j^1|}{|\underline{w}_j^0|+|\underline{w}_j^1|} \frac{1}{|\underline{w}_j^1|} \sum_{\underline{x}_i \in \mathbf{X}^1} \phi(\underline{x}_i)\delta(u_i, j) \\
&= \frac{|\underline{w}_j^0|}{|\underline{w}_j^0|+|\underline{w}_j^1|} \underline{w}_j^0 + (1 - \frac{|\underline{w}_j^0|}{|\underline{w}_j^0|+|\underline{w}_j^1|}) \underline{w}_j^1
\end{aligned} \tag{4.11}$$

We therefore set $\alpha = \frac{|\underline{w}_j^i|}{|\underline{w}_j^i|+|\underline{w}_j|}$ so that, if each mini-batch is labelled correctly at the end of the EM minimization, we retrieve the correct result i.e. same cluster medoids as for full batch kernel k-means.

Empty clusters We close this subsection with a remark about empty-clusters. Indeed it is not guaranteed that along inner loop iterations there will be at least one data sample per each cluster. This is a well known k-means issue and several strategies to deal with such empty-clusters problem are possible e.g. randomly pick a new cluster prototype or reducing N_C.

Here we propose the following: if a given cluster j is found to be empty at the end of the i-th mini-batch iteration then its global prototype will not be updated i.e. $\underline{w}_j \leftarrow \phi(\underline{m}_j^{i-1})$. It is worth noting that this kind of strategy is naturally embedded in the definition of α since for $|\underline{w}_j^i|= 0$ we have $\alpha = 0$ and Eq.4.9 guarantees the correct single batch behavior.

4.2.2 Sparse representation of cluster centroids

In the previous paragraph we introduced a simple yet powerful mini-batch approximation which allowed us to reduce the number of kernel evaluations down to $N\frac{N}{B}$. Here, we show how we can further reduce the complexity of the algorithm by means of an a priori sparse representation of the cluster centroids. As discussed in chapter 2 this approach was first introduced by Chitta et al. [36] in the context of standard kernel k-means. We recall here that such technique relies on the simple observation that the full kernel matrix is required at each iteration of the kernel k-means algorithm because the cluster centers are represented as a linear combination of the entire dataset. However the number of kernel elements to be evaluated can be drastically reduced if one restricts the cluster centers to a smaller sub space spanned by a small number of landmarks i.e. data samples randomly extracted from the dataset.

We illustrate here how we can reformulate the same idea within the framework of our algorithm introducing a sparse representation for cluster centroids at each mini-batch itereation. In order to do so let us recall

Algorithm 1: Mini-batch kernel k-means pseudocode

```
input:  dataset $\mathbf{X}$, number of clusters $N_C$, number of mini-batches $B$
output:  medoids $\mathbf{M}$, labels $\underline{u}$

1  for $i \leftarrow 1$ to $B$ do
2      $\mathbf{X}^i \leftarrow$ samples fetched from $\mathbf{X} \setminus \mathbf{X}^{j<i}$
3      $\mathbf{K}^i \leftarrow$ precompute mini-batch kernel matrix
4      if $i == 1$ then
5          $\mathbf{M} \leftarrow$ initialize according to kernel k-means++
6      end
7      $\underline{u}^i \leftarrow$ assigned according to nearest neighbor medoid among $\mathbf{M}$
8      $t \leftarrow 0$
9      while $\underline{u}^i_t \neq \underline{u}^i_{t+1}$ do
10         $\underline{g}^i \leftarrow$ update according to Eq.4.1
11         $\mathbf{F}^i \leftarrow$ update according to Eq.4.2
12         $\underline{u}^i_{t+1} \leftarrow$ reassign according to Eq.4.3
13         $t \leftarrow t + 1$
14     end
15     $\mathbf{M}^i \leftarrow$ medoid approximation according to Eq.4.4
16     $\mathbf{M} \leftarrow \alpha\mathbf{M} + (1 - \alpha)\mathbf{M}^i$
17 end
18 $\underline{u} \leftarrow$ assigned according to nearest neighbor medoid
```

that while performing kernel k-means on each mini-batch we are implicitly carrying out the following M-step:

$$\underline{w}^i_j \leftarrow \frac{1}{|\underline{w}^i_j|} \sum_{m=1}^{N} \phi(\underline{x}^i_m)\delta(u^i_m, j),\ \forall j \in [1, N_C]$$

Now all we have to do is to restrict the above summation on the subset $m : \underline{x}_m \in L^i$ where $L^i = \{l^i_0, ..., l^i_{|L|}\}$ is a set of $|L|$ landmarks uniformly sampled from the mini-batch.

$$\underline{w}^i_j \leftarrow \frac{1}{|\underline{w}^i_j|} \sum_{m \in L^i}^{N} \phi(\underline{x}^i_m)\delta(u^i_m, j),\ \forall j \in [1, N_C]$$

The update equation for the mini-batch labels will be:

$$u^i_m \leftarrow \arg\min_j [\hat{g}^i_j - 2\hat{F}^i_{(m,j)}] \tag{4.12}$$

where $\hat{\underline{g}}^i$ and $\hat{\mathbf{F}}^i$ are the approximate mini-batch clusters compactness and mini-batch clusters similarity respectively:

$$\hat{g}^i_j = \frac{1}{|\underline{w}^i_j|^2} \sum_{m,n \in L^i} K^i_{m,n} \delta(u^i_m, j) \delta(u^i_n, j) \tag{4.13}$$

$$\hat{F}^i_{m,n} = \frac{1}{|\underline{w}^i_n|} \sum_{l \in L^i} K^i_{m,l} \delta(u^i_l, n) \tag{4.14}$$

It should be clear from Eq. 4.13 and Eq. 4.14 that the number of kernel evaluations needed to run such approximated algorithm is now $Ns\frac{N}{B}$, where the key parameter s is the fraction of data used for the cluster centers representation in each mini-batch defined as:

$$s = \frac{|L|}{N} B \tag{4.15}$$

As already stated in the introduction of the chapter s, together with B, act like knobs that control the degree of approximation of the procedure with respect to standard kernel k-means. In the experimental section we will discuss on how to pick proper values for these parameters according to the available computational resources.

4.3 An efficient distribution strategy

We discuss here how the nature of the previously introduced algorithm is particularly suited to be implemented on systems with a distributed architecture. Let us focus on the distribution strategy for the inner loop of our algorithm analyzing how both data and computations can be effectively scheduled across multiple nodes. We decide here to express the parallelism via a message passing approach, where each node is a peer without a master node.

The primary concern is to decompose the kernel matrix so that the evaluation of its $O(\frac{N}{B})$ elements is evenly distributed across the nodes. Several choices are possible and a few observations about the way these kernel elements are used across the algorithm are needed before detailing our choice. As already discussed in section 4.1, the whole iterative procedure to update the set of predicted labels minimizing the kernel k-means cost function can be expressed in terms of the average cluster similarity $F_i, j, \forall i \in 0, ..., \frac{N}{B}, j \in 0, ..., N_C - 1$ and the cluster compactness $g_j \forall j \in 0, ..., N_C - 1$. Both quantities can be expressed as partial summations of kernel matrix elements, where the elements to be summed are selected according to the labels via $\delta(u_i, j)$. One should note that $\underline{g}$ does not scale with the size of the input whereas $\mathbf{F}$ scales linearly with $\frac{N}{B}$. Therefore, in order to design a proper data distribution pattern, we consider how to scatter the computation of such second quantity minimizing the communication overhead. From

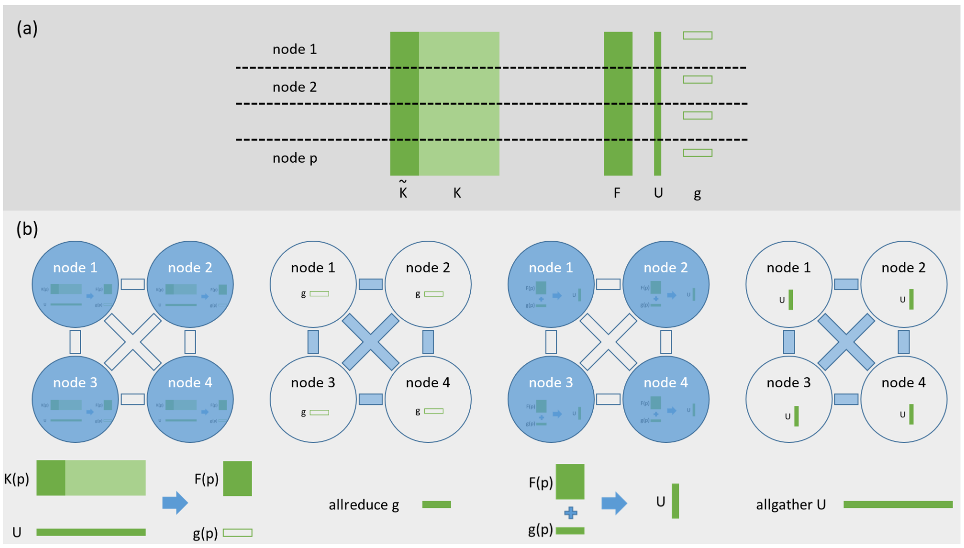

Figure 4.2: (a) Distribution scheme for the principal quantities needed to complete an inner loop iteration. Each node holds a set of entire rows for $\tilde{\mathbf{K}}$, $\mathbf{K}$, $\mathbf{F}$ and $\underline{u}$. Each node holds a local copy of $\underline{g}$ too, however the local information about this vector is partial. The overall information can be retrieved by means of an all-to-all reduction. (b) From left to right the main steps of an inner loop iteration are illustrated. At first, each node is computing its portion of $\mathbf{F}$ together with a partial $\underline{g}(p)$ starting from its $\mathbf{K}(p)$ and $\underline{u}(p)$. Then, the global $\underline{g}$ is retrieved with an all-to-all reduction step. In the third stage each node uses that information together with $\mathbf{F}(p)$ to compute its slice of $\underline{u}$. As a final step an all-to-all gathering step spread the updated labels across the network. At this point it is possible to go on with the next iteration as all the information needed is available to each node. It is worth noting how, along the entire procedure, all the nodes are peers ensuring automatically a good workload balance.

Eq. 4.2 it should be clear that the summation to compute the i-th row of $\mathbf{F}$ runs just over the i-th row of $\mathbf{K}$, this naturally suggests us a row wise distribution strategy. Considering a system with N_P nodes, the workload is divided so that each node p accounts for the computation of $K_{i,j}$ and $F_{i,l}$ $\forall j \in [0, \frac{N}{B}), i \in [p\frac{N}{BN_P}, (p+1)\frac{N}{BN_P}), l \in [0, N_C)$.

The full data distribution scheme is presented in figure 4.2(a) and the resulting algorithm is detailed via pseudo code in Alg.2. The advantage of such approach mainly consists in the reduced communication overhead. Indeed, for each iteration of the inner loop two communication steps are sufficient, involving a reduction of the cluster compactness $\underline{g}$ together with a gathering step for the updated labels $\underline{u}$. The kernel matrix $\mathbf{K}$ as well as the average cluster similarity $\mathbf{F}$ always reside locally to the node and they never go through the network. Per node communications and computations are detailed in figure 4.2(b) and table 4.1.

Algorithm 2: Distributed mini-batch kernel k-means pseudocode for node p

```
input: dataset $\mathbf{X}$, number of clusters $N_C$, number of mini-batches $B$
output: medoids $\mathbf{M}$, labels $\underline{u}$

1  for $i \leftarrow 1$ to $B$ do
2      $\mathbf{X}^i \leftarrow$ samples fetched from $\mathbf{X} \setminus \mathbf{X}^{j<i}$
3      $\mathbf{K}^i(p) \leftarrow$ precompute mini-batch kernel matrix
4      if $i == 1$ then
5          $\mathbf{M} \leftarrow$ initialize according to kernel k-means++
6      end
7      $\underline{u}^i(p) \leftarrow$ assigned according to nearest neighbor medoid among $\mathbf{M}$
8      $t \leftarrow 0$
9      while $\underline{u}^i_t \neq \underline{u}^i_{t+1}$ do
10         allgather $\underline{u}^i_t$                                   // sync
11         $\underline{g}^i(p) \leftarrow$ update according to Eq.4.1
12         $\mathbf{F}^i(p) \leftarrow$ update according to Eq.4.2
13         allreduce sum $\underline{g}^i$                                 // sync
14         $\underline{u}^i_{t+1} \leftarrow$ reassign according to Eq.4.3
15         $t \leftarrow t + 1$
16     end
17     $\mathbf{M}^i(p) \leftarrow$ medoid approximation according to Eq.4.4
18     allreduce min $M_i$                                             // sync
19     $\mathbf{M} \leftarrow \alpha\mathbf{M} + (1-\alpha)\mathbf{M}^i$
20     allreduce min $M$                                               // sync
21 end
22 $\underline{u} \leftarrow$ assigned according to nearest neighbor medoid
```

Algorithm step	Memory	Operations	Comm.
$\mathbf{K}$ evaluation	$\frac{N^2}{B^2 N_P}$	$D\frac{N^2}{B^2 N_P}$	-
$\mathbf{F}$, $\underline{g}$ update	$\frac{N}{BN_P}N_C$	$\frac{N^2}{B^2 N_P}$	N_C
$\underline{u}$ reassignment	$\frac{N}{B}$	$\frac{N}{BN_P}N_C$	$\frac{N}{BN_P}$
medoid approximation	N_C	$\frac{N}{BN_P}N_C$	N_C

Table 4.1: Complexity analysis of the distributed mini-batch algorithm, the factor D introduced in the number of operations for the K evaluatoin accounts for the complexity of the kernel function and the relative distance metric. As an example, the number of operations while using an euclidean based kernel scales linearly also with the dimensionality of the data d.

The memory footprint can be easily computed and amounts to $Q(\frac{N}{BN_P}(\frac{N}{B}+N_C)+\frac{N}{B}+2N_C)$ where Q is the size of variables expressed in Bytes, this is a central quantity because in a real application scenario once fixed the computational resources i.e. amount of memory available per processor R and the number of processors N_P, it allows us to compute the minimum number of mini-batch that can be used to process the entire dataset:

$$B_{\min} = \frac{\frac{2N}{N_P}}{-(\frac{C}{N_P}+1)+\sqrt{(\frac{C}{N_P}+1)^2-8\frac{C}{N_P}+\frac{R}{Q}}} \tag{4.16}$$

An upper bound for the message size per node can also be easily given by $Q(\frac{N}{BN_P}+2N_C)$. This however represents a worst case scenario, where the entire set of labels $\underline{u}$ are communicated at each step, instead of communicating just the ones that were actually updated.

The computational complexity of the proposed implementation grows as $O(\frac{N^2}{BN_P})$ and it is dominated by the kernel matrix evaluation step as it should be clear from table 4.1. It is worth stressing the fact that we decided not to exploit any kernel matrix symmetry because that would have resulted in the impossibility of pursuing our row-wise data distribution scheme and additionally it would have hindered the possibility of using non symmetric similarity functions. Moreover, exploiting the kernel matrix symmetry would have resulted in a non trivial addressing scheme, unsuitable for the limited memory addressing capabilities of accelerators such as general purpose GPUs (gpGPUs). However this increased memory footprint is largely compensated by the approximation strategy in performance terms.

4.4 Discussion

As discussed in the introduction, more specifically in chapter 2, mini-batch approaches are not new in the clustering community and encountered a great success when applied to standard k-means [40]. In his work, Sculley showed how a mini-batch Stochastic Gradient Descent (SGD) procedure converges faster than regular GD. However he proposed to set the size of mini-batches to a rather small value, namely $\approx 10^3$, and to fix an a-priori number of iterations for the algorithm. Our suggestion here is quite different, indeed the number of iterations is by construction equal to the number of mini-batches B in order to exploit the entire dataset. Moreover, a major difference with the SGD procedure proposed by Sculley is here represented by the inner loop. We actually believe that iterating each mini-batch up to convergence can lead to a better minimization of the cost function and to a less noisy procedure. To prove this point in the experimental section the reader can find a comparison between the here proposed algorithm and the mini-batch SGD procedure proposed by Sculley.

We stress also the fact that our parallelization approach is rather different when compared to parallel patch clustering algorithms [42] as they were discussed in the introductory chapter. Indeed, we don't parallelize across mini-batches assigning one mini-batch per node. Instead, we parallelize the iterations within each mini-batch thus allowing the algorithm to better cope with large sample size. In Fig.4.3 one can better appreciate such difference in a scenario where the available computational resources are limited with respect to the size of $\mathbf{K}$. In such realistic case of application it is evident how our algorithm provides a better approximation to the single-batch result. Indeed the DKK algorithm, being able to distribute the $\mathbf{K}^i$ partial kernel matrix over the entire network of nodes can cope with significantly larger mini-batches of the data per iteration.

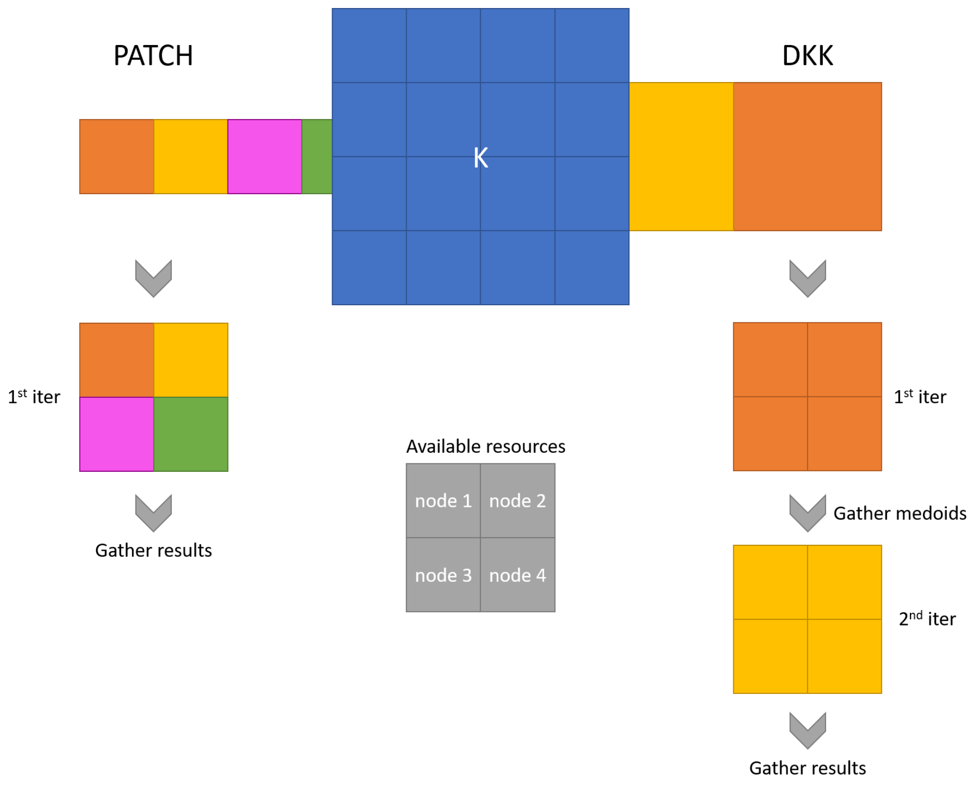

Figure 4.3: Graphical representation of the two different mini-batch parallelization approaches of Patch clustering and DKK. A system of 4 nodes with limited amount of memory per node R_n (represented as the area of the central gray squares) is taken into consideration with respect to a kernel matrix $\mathbf{K}$ (in blue) that requires $16R_n$ available memory. In order to cope with such matrix the Patch algorithm requires a subdivision of the data into 4 small mini-batches and runs in a single trivially parallel iteration. The proposed DKK algorithm when dealing with the same matrix is able to gather a global results in two iterations, requiring however just 2 larger mini-batches. The quality of the global clustering result is expected to be higher for the DKK algorithm since it take into consideration $\frac{N^2}{2}$ kernel elements whereas the patch algorithm take into consideration a smaller fraction of them i.e. $\frac{N^2}{4}$

Chapter 5

GPU Accelerated DKK for Clustering MD Data

In the previous chapter we introduced a novel clustering engine, namely Distributed Kernel K-means (DKK). We showed how starting from a proper reformulation of the original kernel k-means algorithm is possible to devise an efficient distribution strategy that together with the proposed two-fold approximation is able to reduce dramatically the computational burden and the memory footprint of exact kernel methods. It is worth observing however that, even if the number of kernel evaluations in the proposed DKK algorithm is cut down to $N \times s\frac{N}{B}$, the performances still depend heavily on the actual implementation of such kernel matrix evaluation step. This observation is particularly relevant if one desires to perform clustering over molecular dynamics trajectories, indeed as we discussed in Chapter 2 such scenario require a similarity measure based on Minimum Root Min Square Displacement e.g. gaussian kernel in the form of:

$$K(\underline{x}_i, \underline{x}_j) = e^{-\frac{\mathrm{RMSD}(\underline{x}_i, \underline{x}_j)^2}{\sigma^2}}$$

The evaluation step of such kind of complex kernel matrix or sub-matrix can represents a major bottleneck in the proposed algorithm affecting the overall performances.

In this chapter we tackle the problem proposing an accelerated version of the DKK algorithm which is able to run on *state-of-the-art* heterogeneous computational platforms. At first we discuss an offload acceleration strategy carefully designed to exploit the iterative nature of the mini-batch algorithm. Even though such acceleration strategy can be implemented in principle for all sort of offload accelerators (general purpose GPUs (gpGPUs) as well as Intel Many Integrated Core architectures) we focus the attention in the second part of the chapter on the design of an actual CUDA implementation for nVIDIA Graphic Processing Units (GPUs). The reason for this choice has to be found in both the popularity of such accelerators on the market and on the great control that CUDA offers to the developer allowing low-level memory optimizations.

5.1 Offload acceleration strategy

In the following section we discuss how the mini-batch structure of the algorithm can be exploited in order to design an effective acceleration strategy. We will consider an offload acceleration model where the host processor and the target device have separate memory address spaces and communicate via a bus with limited bandwidth (e.g. PCIe) with respect to the processor-memory standard bus.

As already discussed the bottleneck of the computation in real application scenarios is usually the kernel matrix evaluatoin. The evaluation of such large kernel matrix or sub-matrix perfectly fits the massively parallel architecture of nowadays accelerators, therefore it is a reasonable choice to offload that portion of the computation. One of the key elements for an efficient acceleration scheme is the overlapping in time between the host and the target workload [63].

There is no hope to find an overlapping scheme within the inner loop of the algorithm since the entire procedure depends on the kernel matrix elements to be completed, therefore we concentrate our efforts on the outer loop. Each i-th iteration depends on the previous ones, in order to initialize the set of labels $\underline{u}^i$. This is what prevents the algorithm to be trivially parallel forcing to run just one mini-batch per time. However if one considers the first two steps of each outer loop iteration i.e. mini-batch fetch $\mathbf{X}^i$ and kernel matrix evaluation $\mathbf{K}^i$ it is clear that they can be performed independently for each i. We exploit this feature, instructing the target device to compute the kernel matrix $\mathbf{K}^{(i+1)}$ while the host processor executes the inner loop of the algorithm on the i-th mini-batch.

The offload procedure is detailed in Fig. 5.1. The overall performance gain for such acceleration strategy however heavily depends on the accelerator side implementation of the kernel matrix evaluation. For this reason the following section deals with the details of a CUDA implementation for the Root Mean Square Deviation (RMSD) kernel matrix evaluation.

5.2 Fast RMSD kernel evaluation with CUDA

Hereafter we present an efficient many-core implementation for the kernel matrix evaluation in case of a Gaussian kernel with optimal RMSD metric.

First, we briefly introduce the quaternion-based algorithm (QCP) originally developed by Theobald in [64] to compute the minimum RMSD between two conformations. An efficient and original many-core implementation is then discussed. With this respect we focused our efforts on the design

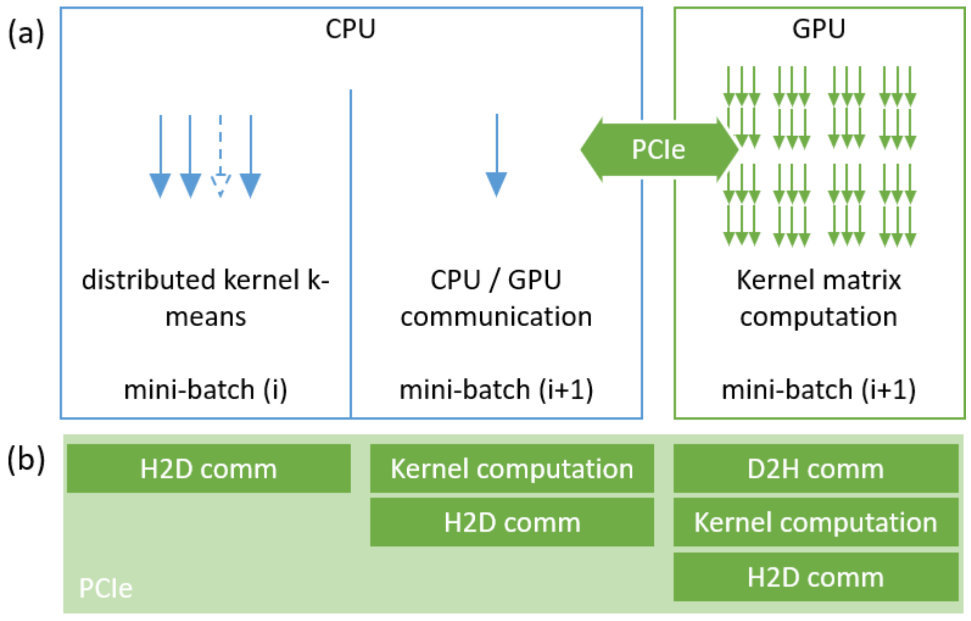

Figure 5.1: (a) Pictorial description of the proposed acceleration scheme. The diagram is divided in two parts: a host processor side on the left, and a target device side on the right. We illustrate how multiple CPU threads can be used to overlap host and device workload. A CPU thread is bound to the device, it is responsible for data fetching from disk, for host-device data transfer and for device control. It instructs the device to compute the kernel matrix elements needed by the next $i+1$-th iteration of the outer loop. All the other available threads cooperate and are responsible for the current i-th iteration consuming the kernel matrix elements provided by the accelerator. In this sense device and host work in a producer-consumer pattern. (b) We detailed how a 3-stage pipeline can be used on the device in order to overlap the kernel computation with the host to device (H2D) and device to host (D2H) slow communications needed to transfer the dataset on the device and the kernel matrix back to host respectively.

of ad-hoc memory layout for data structures enhancing memory coalescence.

5.2.1 The QCP algorithm for minimum RMSD

Let us consider two conformations $\mathbf{A}$ and $\mathbf{B}$ of a macromolecule made of N_a atoms. Each conformation is represented by a $N_a \times 3$ coordinate matrix i.e. $A_{i,\cdot} = (x_i^A, y_i^A, z_i^A); B_{i,\cdot} = (x_i^B, y_i^B, z_i^B)\ \forall i \in [0, N_a - 1]$. For sake of simplicity, without loss of generality, we assume that each conformation is self-centered, i.e. translated into the origin. We introduce here three central quantities for the quaternion-based algorithm:

$$\mathbf{S}^{i,j} = {A_{i,\cdot}}^T B_{j,\cdot} = \begin{bmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{bmatrix}^{i,j} \tag{5.1}$$

$$G_A = \mathrm{Tr}(\mathbf{A}^T\mathbf{A}) \tag{5.2}$$

$$G_B = \mathrm{Tr}(\mathbf{B}^T\mathbf{B}) \tag{5.3}$$

Minimum RMSD can be written as a minimization problem of the mean square error over the set of orthogonal rotations $\mathbf{R}$:

$$\mathrm{RMSD}^2 = \min_{\mathbf{R}} \frac{\|\mathbf{A} - \mathbf{BR}\|_F^2}{N_a}$$

It can be shown [64] that such minimization problem can be solved looking for the largest positive eigenvalue λ_M of:

$$\mathbf{H} = \begin{bmatrix} S_{xx}+S_{yy}+S_{zz} & S_{yz}-S_{zy} & S_{zx}-S_{xz} & S_{xy}-S_{yx} \\ S_{yz}-S_{zy} & S_{xx}-S_{yy}-S_{zz} & S_{xy}+S_{yx} & S_{zx}+S_{xz} \\ S_{zx}-S_{xz} & S_{xy}+S_{yx} & -S_{xx}+S_{yy}-S_{zz} & S_{yz}+S_{zy} \\ S_{xy}-S_{yx} & S_{zx}+S_{xz} & S_{yz}+S_{zy} & -S_{xx}-S_{yy}+S_{zz} \end{bmatrix}$$

which can be found looking for the largest positive root of the characteristic equation:

$$P(\lambda) = \lambda^4 - 2\,\mathrm{Tr}(\mathbf{S}^T\mathbf{S})\lambda^2 - 8|\mathbf{S}|\lambda + |\mathbf{H}| = 0 \tag{5.4}$$

As suggested by Theobald, the Newton-Raphson iterative method can be effectively used to obtain a numerically stable solution for such problem. The eigenvector corresponding to λ_M is a quaternion equivalent to the optimal rotation and the RMSD can be computed as:

$$\text{RMSD} = \sqrt{\frac{G_A + G_B - 2\lambda_M}{N_a}} \tag{5.5}$$

It is worth observing that since all the coefficients in Eq.5.4 depend on the elements of the matrix S the entire procedure can be carried out with few FLoating point OPerations (FLOP) once that the summations involved in the evaluation of S, G^A and G^B are computed.

5.2.2 Design of an effective CUDA implementation

The kind of computational platform we are considering is described in Fig.5.2: a many-core architecture where cores are grouped into multiprocessors featuring a hierarchical memory structure. A software abstraction layer such as CUDA allows us to express parallelism for such a machine in a Single Instruction Multiple Threads (SIMT) fashion i.e. each core within a given multiprocessor executes the same instruction on a different subset of the data selected according to the executing thread index. We assume the threads to be organized in a 2D blocks grid formed by square blocks of $B \times B$ threads. A runtime scheduler will be responsible for the threads scheduling so that threads belonging to the same block are executed together on the same multiprocessor.

We assume one control unit per multiprocessor. According to the SIMT execution model, the control unit fetches one operation and instructs all the cores within the multiprocessor to execute it on their own operands. Therefore, avoiding execution branches within a thread block should be a major concern in order to avoid code serialization.

We assume also that load/store operations on global memory are strongly affected by the access pattern i.e. threads within a block should address contiguous portions of global memory in order to perform a load/store operation in a single transaction. This is exactly the case for general purpose GPU architectures therefore the design of a proper memory layout for the data structures is of paramount importance.

In order to design a proper data structure and in agreement with CUDA specifications we finally assume that vector types are available together with optimized vector instructions e.g. two float variables can be stored as a single float2 variable in order to take advantage of a single vectorized load/store instruction.

Let now consider a set of conformations $\mathbf{X} = \{\mathbf{X}_i\} \forall i \in [0, N-1]$, we are interested in computing the Gaussian kernel matrix with optimal RMSD metric on such dataset i.e. $K_{i,j} = \exp - \frac{\text{RMSD}(\mathbf{X}_i, \mathbf{X}_j)^2}{\sigma^2}$. Since the evaluation of a single matrix element with the previously introduced QCP algorithm is computationally cheap we expose parallelism by computing more than one kernel matrix elements per time. We designed an implementation where each thread is responsible for the evaluation of a single entry in the kernel

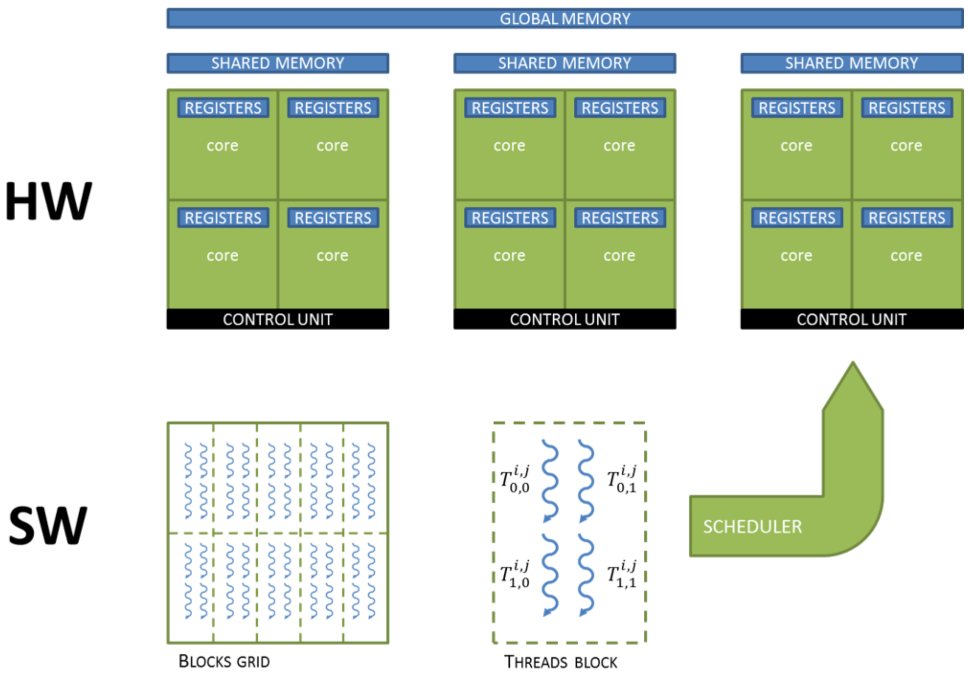

Figure 5.2: Simple block diagram for a many-core computational platform hardware together with a pictorial description of a software abstraction layer exposing SIMT parallelism. A runtime scheduler is responsible for the mapping of the thread blocks onto the hardware multiprocessors. More than one thread block can be scheduled to run concurrently on a single multiprocessor in order to hide memory latency. The hierarchical memory structure is illustrated, from top to bottom the memory size reduces and the bandwidth increases.

matrix but all the threads within a block collaborate to load the needed data into shared memory in order to guarantee coalesced memory access to global memory. We recall that each sample $\mathbf{X}_i$ is represented by a $N_a \times 3$ matrix containing the coordinates of each atom. In the following the coordinate of the generic j-th atom will be indicated as (x_j^i, y_j^i, z_j^i) so that a subscript represent the atom index whereas a superscript stands for the sample index.

As shown in Fig.5.2, let $T_{l,m}^{i,j}$ be the thread (l, m) within the block (i, j) of our blocks grid. Such thread will be responsible for the evaluation of the kernel element $K_{iB+l,jB+m}$. As described above in order to carry out such computation the following intermediate quantities need to be computed:

$$\mathbf{S}^{iB+l,jB+m} = (\mathbf{X}_{iB+l})^T \mathbf{X}_{jB+m} \tag{5.6}$$

$$G_i = \mathrm{Tr}(\mathbf{X}_i^T \mathbf{X}_i) \tag{5.7}$$

$$G_j = \mathrm{Tr}(\mathbf{X}_j^T \mathbf{X}_j) \tag{5.8}$$

Therefore we have to store in the block shared memory the atoms of the conformations $\mathbf{X}_n\ \forall n \in [iB, i(B+1)-1] \cup [jB, j(B+1)-1]$. Assuming 4 bytes per float this accounts for $4 \times 6BN_a$ bytes to be stored in shared memory. Being the size of shared memory limited, storing all these variables can dramatically affect performances. Specifically, the more resources each block is demanding, the less blocks can be scheduled concurrently on each multiprocessor to mask memory latency. In order to overcome such problem, we propose to divide each frame into atom chunks of size fB where f represents the size of the vector type used. The atom chunks can be processed one after the other lowering the amount of shared memory needed per block to $4 \times 6fB^2$.

Let us now discuss how data can be arranged so as to have as much coalesced memory accesses as possible. Each block of threads need to loop over atom chunks for two different frame chunks i.e. frames $[iB, i(B+1)-1]$ and frames $[jB, j(B+1)-1]$. In Fig.5.3, we illustrate an hierarchical memory layout that exploits such information to allow the retrieval of an atom chunk for the needed frames in 6 perfectly coalesced load instructions.

Once an atom chunk for the needed frames is loaded into the shared memory, each thread of the block can update its current quantities as follows:

$$S_{\star,\bullet}^{iB+l,jB+m} \leftarrow S_{\star,\bullet}^{iB+l,jB+m} + \sum_{n=a}^{a+fB-1} \star_n^{iB+l} \bullet_n^{jB+m}; \quad \star,\bullet = [x,y,z] \tag{5.9}$$

$$G_i \leftarrow G_i + \sum_{n=a}^{a+fB-1} (x_n^i)^2 + (y_n^i)^2 + (z_n^i)^2 \tag{5.10}$$

$$G_j \leftarrow G_j + \sum_{n=a}^{a+fB-1} (x_n^j)^2 + (y_n^j)^2 + (z_n^j)^2 \tag{5.11}$$

In Fig.5.4 we show how with a simple rotational index such update can be performed avoiding shared memory bank conflicts among the block threads.

At this stage all the main quantities needed to perform the minimum RMSD computation are available and each thread can straightforwardly perform the rest of the computation using its own registers. The entire procedure is detailed in Alg.3. It is worth noting that to avoid execution branches the Newton-Raphson method is implemented with a fixed number of iterations. We implemented the proposed algorithm in CUDA and we tested it in a real application scenario as it will be described in the experimental part of the thesis i.e. chapter 7.

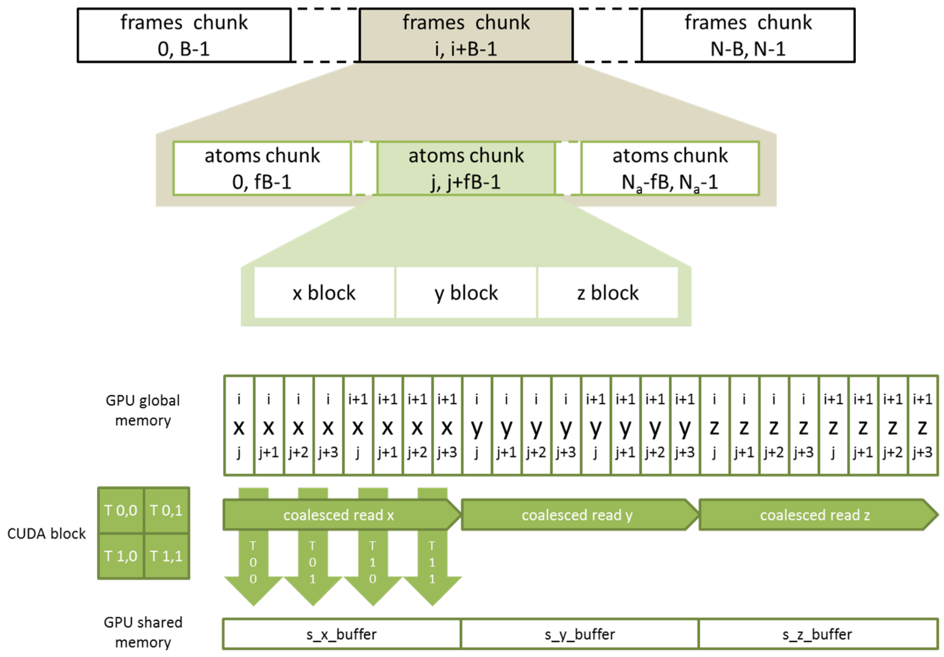

Figure 5.3: Hierarchical memory layout for a set of frame conformations designed in order to achieve memory coalescent access while performing an RMSD matrix computation. We recall that superscripts refer to frame indices whereas subscripts refer to atom indices within a frame. (top) Block diagram illustrating the layout. Frames are stored in frame chunks of size B i.e. each frame chunk contains all the N_a atoms of B consecutive frames. The atoms themselves are stored in chunks of size fB i.e. each atom chunk contains the same set of fB atoms for all the B frames. Finally, the atom coordinates are stored as three consecutive blocks for the x, y and z components. (bottom) Detailed view of one atom chunk for $B = 2$ and $f = 2$ together with a pictorial description of the three coalescent load instructions needed to read it.

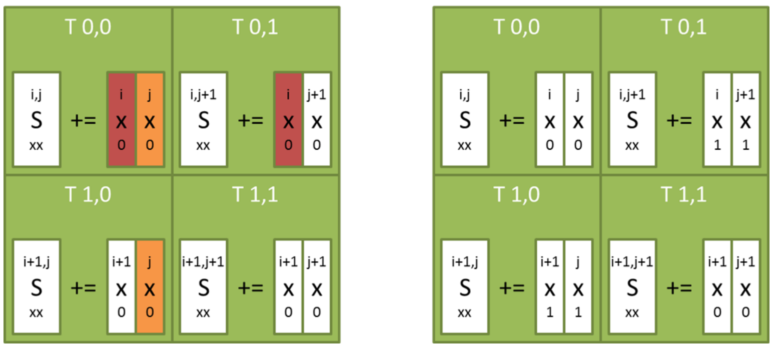
T 0,0
T 0,1
T 1,0
T 1,1
i,j S xx += i x 0 j x 0
i,j+1 S xx += i x 0 j+1 x 0
i+1,j S xx += i+1 x 0 j x 0
i+1,j+1 S xx += i+1 x 0 j+1 x 0
i,j S xx += i x 0 j x 0
i,j+1 S xx += i x 1 j+1 x 1
i+1,j S xx += i+1 x 1 j x 1
i+1,j+1 S xx += i+1 x 0 j+1 x 0

Figure 5.4: Example showing how to implement a rotational index to avoid shared memory bank conflicts while computing the summations described in Eq.5.9, 5.10 and 5.11. We recall that superscripts refer to frame indices whereas subscripts refer to atom indices within a frame. (left) A straightforward implementation of the summation will cause all the threads to work on the same atom at each iteration. This cause shared memory bank conflicts that strongly affect performances. (right) A simple rotational atom index is used so that each thread work on a different atom while computing the summation: ridx = n + tIdx.x + tIdx.y%B.

5.3 Implementation parameters

A final remark has to be done on the implementation parameters i.e. f and B. Tuning such parameters depending on the hardware is crucial in order to get peak performances. Among the two, choosing f is simpler: looking at the specifications of the hardware one can verify the existence of a vectorized load instruction and choose f accordingly (e.g. if the hardware provide a 64-bit vectorized load instruction one can safely set $f = 2$). In order to properly choose the block size B we rather suggest to look at two different hardware properties: the available shared memory and the number of threads scheduled together W (i.e. warp size for nVidia GPUs). We obviously want the number of threads per block B^2 to match an integer multiple of W and we would like to reduce the amount of memory required by each block in order to increase the relative occupancy of the multiprocessors.

$$B^2 = iW, \quad i > 1 \tag{5.12}$$

$$4 \times 6fB^2 << M \tag{5.13}$$

For instance, in the experimental section later discussed where an nVidia GTX980 board is used, we set $f = 2$ since an 8-byte vectorized load instruction is available and $B = 8$ requiring 3kB of shared memory per block over the available 96kB.

5.4 Discussion

We close this chapter commenting on the fact that nVIDIA GPUs are by far the most common many-core accelerators on the market and that all major Molecular Dynamics (MD) simulations software tools [65, 66], are nowadays offering a CUDA accelerated version of their code. It should be therefore clear the importance for post-processing tools to be able to run natively on GPU accelerated architectures. The proposed DKK algorithm together with the accelerated implementation presented in this chapter perfectly fulfill this requirement.

It is also worth stressing the fact that up to our knowledge the low level MD data structure optimization proposed here is novel and represents an advancment with respect to other proposed CUDA RMSD implementations e.g. [67].

Algorithm 3: Gaussian kernel evaluation with minimum RMSD metric, pseudocode for a many core implementation.

```
input:
dataset $\mathbf{X}$, number of atoms $N_a$, block size $B$, vector type size $f$
block index bIdx, thread index: tIdx
register variables:
$S_{xx}$, $S_{xy}$, $S_{xz}$, $S_{yx}$, $S_{yy}$, $S_{yz}$, $S_{zx}$, $S_{zy}$, $S_{zz}$, $G_i$, $G_j$
shared memory variables:
buffers for frames $[iB, i(B+1)-1]$: $\underline{\text{ix}}$, $\underline{\text{iy}}$, $\underline{\text{iz}}$
buffers for frames $[jB, j(B+1)-1]$: $\underline{\text{jx}}$, $\underline{\text{jy}}$, $\underline{\text{jz}}$
output:
Kernel matrix $\mathbf{K}$

tID = tIdx.y*$B$+tIdx.x
offi = bIdx.y * $3BN_a/f$
offj = bIdx.x * $3BN_a/f$
for $r \leftarrow 0, \frac{N_a}{fB}$ do
    $\underline{\text{ix}}$[tID] = $\mathbf{X}$[offi+tID]
    $\underline{\text{iy}}$[tID] = $\mathbf{X}$[offi+B*B+tID]
    $\underline{\text{iz}}$[tID] = $\mathbf{X}$[offi+2*B*B+tID]
    load in the same way $\underline{\text{jx}}$, $\underline{\text{jy}}$, jz
    offi += B*B*3
    offj += B*B*3
    sync block threads
    for $n \leftarrow 0, fB-1$ do
        r = (n+tIdx.x+tIdx.y)%$B$
        $S_{xx}$ += $\underline{\text{ix}}$[tIdx.y*$B$+r]*$\underline{\text{jx}}$[tIdx.x*$B$+r]
        update in the same way $S_{xy}, S_{xz}, S_{yx}, S_{yy}, S_{yz}, S_{zx}, S_{zy}, S_{zz}$
        $G_i$ = $G_i$ + $\underline{\text{ix}}$[tIdx.y*$B$+r]$^2$+$\underline{\text{iy}}$[tIdx.y*$B$+r]$^2$+$\underline{\text{iz}}$[tIdx.y*$B$+r]$^2$
        $G_j$ = $G_j$ + $\underline{\text{jx}}$[tIdx.x*$B$+r]$^2$+$\underline{\text{jy}}$[tIdx.x*$B$+r]$^2$+$\underline{\text{jz}}$[tIdx.x*$B$+r]$^2$
    end
    sync block threads
end
compute c0, c1, c2, c3, c4 according to Eq.5.4
for $iterations \leftarrow 0, MAX\_ITERATIONS$ do
    $\lambda$ -= (c4*$\lambda^4$ + c3*$\lambda^3$ + c2*$\lambda^2$ + c1*$\lambda$ + c0)/(4*c4*$\lambda^3$ + 3*c3*$\lambda^2$
      + 2*c2*$\lambda$ + c1)
end
msd = ($G_i$+$G_j$-2*$\lambda$)/$N_a$
K[bIdx.y*B+tIdx.y,bIdx.x*B+tIdx.x] = exp(-msd/$\sigma^2$)
```

Chapter 6

A Principal Paths Finding Algorithm in Kernel Space

The previous two chapters introduced two rather technological advancements in the field of Unsupervised Learning (UL) for Molecular Dynamics (MD). Both the Distributed Kernel K-means (DKK) algorithm and its accelerated version were developed following the requirements of the clustering problems applied to MD datasets i.e. to large datasets not embeddable in a vector space requiring an expensive distance matrix evaluation. In chapter 3 a further connection between MD and UL was highlighted: there we showed how MD not only pushes the technological aspects of UL as an increasingly demanding domain of application but is also able to inspire totally new UL principles.

With this respect in the same chapter, starting from the concept of Minimum Free Energy Path (MFEP) we introduced the one of Principal Paths in Data Space as a natural solution to the problem of inferring a smooth path connecting a starting sample to an ending one, locally passing through the *middle* of the data.

Hereafter we show how a kernel based method can be derived in order to approximate such kind of paths. We introduce a novel regularized cost function starting from the standard kernel k-means cost by addition of a 1D topology imposed as a set of harmonic restraints together with fixed boundary conditions (i.e. fixed starting and ending cluster centers). The minimization of such cost naturally leads to an Expectation Maximization (EM) algorithm that will be discussed both in the original and in a kernel space. From an algorithmical standpoint one may think at a Principal Path as a regularized path, with fixed boundaries, attracted towards the local center of mass of the data.

As it will be clear, the quality and the smoothness of the solution found by such algorithm is ruled by the regularization parameter. Informally we can think to this central parameter as the one that controls the trade-off between the path smoothness and how much the path *passes through* the data. A large portion of the chapter is therefore dedicated to the model selection phase of such parameter. More specifically, a Bayesian maximal evidence principle that allows blindfolded in sample model selection is derived.

Throughout the chapter the following notation will be used:

- N is the number of samples.
- N_C is the number of cluster prototypes.
- $\phi(\cdot) : \mathbb{R}^d \to \mathbb{R}^{d'}$ is the possibly non-linear transformation mapping the d-dimensional input space into a d'-dimensional transformed one.
- $\mathbf{X}$ is the $N \times d$ matrix of samples x_i arranged in a row wise fashion i.e. $\mathbf{X}_{i,\cdot} = \underline{x_i}$.
- $\mathbf{K}$ is the $N \times N$ kernel matrix defined as $\mathbf{K}_{i,j} = \langle \phi(\underline{x}_i), \phi(\underline{x}_j) \rangle$.
- $\mathbf{W}$ is the $N_C \times d'$ matrix of cluster prototypes $\underline{w}_i$ arranged in a row wise fashion.
- $|w_i|$ represents the cardinality of the i-th cluster.
- $u_i \in [1, N_C]$ is the label associated with the i-th sample $\underline{x}_i$.
- $\underline{w}_0$ and $\underline{w}_{N_C+1}$ represents the boundary conditions of the algorithm i.e. the starting and ending points of the inferred path.

6.1 The cost function

We now give a formal definition of the newly defined principal path learning problem by introducing a regularized cost function to be optimized. As usual in the context of regularized functional optimization we write a cost having the following form:

$$\Omega(\mathbf{W}, \underline{u}, \mathbf{X}, \gamma, \lambda) = \gamma \Omega_X(\mathbf{W}, \underline{u}, \mathbf{X}) + \lambda \Omega_W(\mathbf{W}) \tag{6.1}$$

More specifically, the primal problem that we aim to optimize in order to infer a smooth transition path from the starting point $\underline{w}_0$ to the ending point $\underline{w}_{N_C+1}$ is the following:

$$\min_{\mathbf{W}} \frac{\gamma}{2} \sum_{i=1}^{N} \sum_{j=1}^{N_C} \|\phi(\underline{x}_i) - \underline{w}_j\|^2 \delta(u_i, j) + \frac{\lambda}{2} \sum_{i=0}^{N_C} \|\underline{w}_{i+1} - \underline{w}_i\|^2 \tag{6.2}$$

As anticipated, this functional represents a regularized version of the standard kernel k-means cost already discussed in chapter 2. Here the first and last clusters, namely $\underline{w}_0$ and $\underline{w}_{N_C+1}$ are kept fixed as boundary conditions

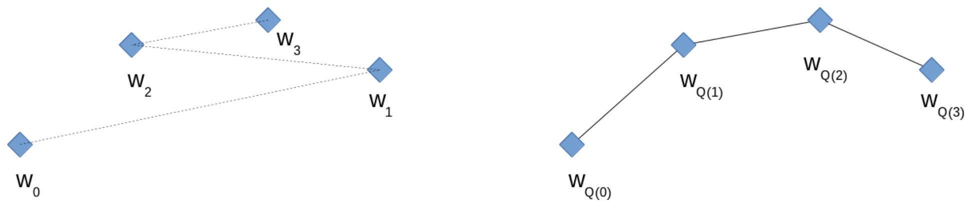

Figure 6.1: An example of index permutation Q to initialize a 1D topology i.e. an N_C-segments curve. The initial cluster centers can be picked with standard k-means initialization algorithms such as k-means++ while the permutation Q can be derived by some rational such as for example a shortest path algorithm on top of a fully connected topology.

and a 1D topology is forced by the proposed quadratic regularization cost Ω_W which introduces a set of harmonic restraints connecting subsequent cluster prototypes $(\underline{w}_{i+1}, \underline{w}_i)$.

The resulting one-dimensional topology is assumed to be set beforehand by means of a simple permutation $Q : \mathbb{Z}^+_{N_C} \to \mathbb{Z}^+_{N_C}$ of the cluster indices as detailed in Fig.6.1. For the sake of convenience in the following, without loss of generality we will assume Q to be the identity so that $Q(i) = i$.

We stress the fact that, as in the case of kernel k-means, Ω is a non-convex function with respect to $\mathbf{W}$ mainly because of the hard-assignment of latent labels i.e. $\delta(u_i, j)$. The two hyperparameters γ and λ regulate the trade-off between data-fitting and smoothness of the inferred path as shown in Fig.6.2(a). It is worth noting that for the sole purpose of optimization, only the ratio $s = \frac{\lambda}{\gamma}$ is relevant, being γ a simple scaling factor. Hereafter we will refer to s as regularization parameter; why keeping γ and λ separated during the derivation will be clear later when a Bayesian model selection framework will be introduced.

Lemma 1. *The primal cost function has the following compact trace formulation:*

$$\begin{aligned}
&\Omega(\mathbf{W}, \underline{u}, \mathbf{X}, \gamma, \lambda) \\
&= \frac{\gamma}{2}\operatorname{Tr}(\mathbf{W}^T\mathbf{A}_\mathbf{X}(\underline{u})\mathbf{W} - 2\mathbf{C}^T\mathbf{W} + \mathbf{D}_\mathbf{X}) + \\
&+ \frac{\lambda}{2}\operatorname{Tr}(\mathbf{W}^T\mathbf{A}_\mathbf{W}\mathbf{W} - 2\mathbf{B}^T\mathbf{W} + \mathbf{D}_\mathbf{W})
\end{aligned} \tag{6.3}$$

Proof. Eq.6.3 is derived by construction, having defined the following matrices:

- The $N_C \times d'$ boundary condition matrix:

$$\mathbf{B}_{i,\cdot} = \begin{cases} \underline{w}_0 \text{ if } i = 1 \\ \underline{w}_{N_C+1} \text{ if } i = N_C \\ \underline{0} \text{ othrewise} \end{cases}$$

- The $N_C \times d'$ centroid matrix: $\mathbf{C}_{i,\cdot} = \sum_{j=1}^{N} \phi(\underline{x}_j)\delta(u_i, j)$, where $\delta(u_i, j)$ is the Kronecker delta.

- The $N_C \times N_C$ hessian matrix of the standard k-means cost function i.e. $\mathbf{A}_{\mathbf{X}i,j} = \begin{cases} |w_i|, \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}$.

- The $N_C \times N_C$ Toeplitz matrix:

$$\mathbf{A}_{\mathbf{W}} = \begin{bmatrix} 1 & -\frac{1}{2} & 0 & \dots & 0 \\ -\frac{1}{2} & 1 & -\frac{1}{2} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\frac{1}{2} & 1 & -\frac{1}{2} \\ 0 & \dots & 0 & -\frac{1}{2} & 1 \end{bmatrix}$$

- The $d' \times d'$ matrix $\mathbf{D}_{\mathbf{X}} = \mathbf{D} + \mathbf{B}^T\mathbf{A}_{\mathbf{B}}\mathbf{B} - 2\mathbf{B}^T\mathbf{C}_{\mathbf{B}}$ where

$$\mathbf{D}_{i,j} = \sum_k \phi(\underline{x}_k)_i \phi(\underline{x}_k)_j$$

,

$$\mathbf{A}_{\mathbf{B}i,j} = \begin{cases} |w_0|, \text{ if } i = 1 \\ |w_{N_C+1}|, \text{ if } i = N_C \\ 0 \text{ otherwise} \end{cases}$$

$$\mathbf{C}_{\mathbf{B}i,\cdot} = \begin{cases} \sum_{j=1}^{N} \phi(\underline{x}_j)\delta(u_i, 0), \text{ if } i = 1 \\ \sum_{j=1}^{N} \phi(\underline{x}_j)\delta(u_i, N_C + 1), \text{ if } i = N_C \\ 0 \text{ otherwise} \end{cases}$$

- The $d' \times d'$ matrix $\mathbf{D}_{\mathbf{W}} = \mathbf{B}^T\mathbf{B}$

$\square$

6.2 EM optimization

We now derive an optimization algorithm for Eq.6.3 in the spirit of the EM procedure originally proposed by Bottou and Bengio [32] in the context of k-means clustering. In the following we assume the transformation $\phi(\cdot) : \mathbb{R}^d \to \mathbb{R}^{d'}$ to be either explicitly known or not present at all. In the next section we will then prove that the derived algorithm can be carried out with the only notion of the kernel matrix $\mathbf{K}$.

E-step assuming a set $\mathbf{W}_t$ of prototypes at a given iteration t, we set the latent labels $\underline{u}$ to be equal to the one minimizing $\Omega(\mathbf{W}_t, \underline{u}, \mathbf{X}, \gamma, \lambda)$ i.e. minimizing $\Omega_X(\mathbf{W}_t, \underline{u}, \mathbf{X})$ since $\Omega_W(\mathbf{W}_t)$ does not depend on $\underline{u}$:

$$u_{i,t+1} \leftarrow \arg\min_j \|\phi(\underline{x}_i) - \underline{w}_{j,t}\|^2 \tag{6.4}$$

We stress the fact that this is exactly the E-step usually performed in a standard kernel k-means implementation.

M-step having computed the updated labels $\underline{u}_{t+1}$ we now minimize Ω with respect to the set of prototypes $\mathbf{W}$ keeping $\underline{u}_{t+1}$ fixed. Computing the partial derivative and setting it to zero

$$\frac{\partial\Omega(\mathbf{W}, \underline{u}_{t+1}, \mathbf{X}, \gamma, \lambda)}{\partial\mathbf{W}} = (\gamma\mathbf{A}_{\mathbf{X}}(\underline{u}_{t+1}) + \lambda\mathbf{A}_{\mathbf{W}})\mathbf{W} - (\gamma\mathbf{C} + \lambda\mathbf{B}) = 0$$

we obtain:

$$(\gamma\mathbf{A}_{\mathbf{X}}(\underline{u}_{t+1}) + \lambda\mathbf{A}_{\mathbf{W}})\mathbf{W} = \gamma\mathbf{C} + \lambda\mathbf{B} \tag{6.5}$$

.

This is a linear system of the Toeplitz type in $d'N_C$ unknowns that need to be solved at each iteration of the algorithm. The system can be solved exactly by Levinson-Durbin recursion method [68] in $\Theta((d'NC)^2)$ time. In the following sections we will also explore some approximated techniques useful in those cases for which the computational cost of an exact solver is prohibitive. Hereafter, without loss of generality, we assume to solve the given system by inversion:

$$\mathbf{W} = (\gamma\mathbf{A}_{\mathbf{X}}(\underline{u}_{t+1}) + \lambda\mathbf{A}_{\mathbf{W}})^{-1}(\gamma\mathbf{C} + \lambda\mathbf{B}) \tag{6.6}$$

thus obtaining the following update equation for the cluster prototypes:

$$\mathbf{W}_{t+1} \leftarrow (\gamma\mathbf{A}_{\mathbf{X}}(\underline{u}_{t+1}) + \lambda\mathbf{A}_{\mathbf{W}})^{-1}(\gamma\mathbf{C} + \lambda\mathbf{B}) \tag{6.7}$$

The following lemma proves the invertibility of the matrix $\mathbf{A}$.

Lemma 2. *The matrix* $\mathbf{A} = \gamma\mathbf{A}_{\mathbf{X}} + \lambda\mathbf{A}_{\mathbf{W}}$ *is invertible.*

Proof. Being $\mathbf{A}$ a tridiagonal matrix with $\mathbf{A}_{i,i} = \gamma|\underline{w}_i|+\lambda$ and with off-diagonal elements $\mathbf{A}_i, j = -\frac{\lambda}{2}$, its eigenvalues μ_i are bounded by the well known Gershgorin Circle Theorem:

$$\gamma|w_i|\leq \mu_i \leq \gamma|w_i|+2\lambda$$

Assuming non-empty clusters i.e. $|w_i|\geq 0$ and recalling that $\lambda \geq 0$ this result shows that in non-trivial cases i.e. $\gamma > 0$, the matrix $\mathbf{A}$ is positive-definite therefore is also invertible. □

6.3 EM optimization algorithm in kernel space

We now show how the EM algorithm introduced in the previous section can be carried out without an explicit notion of the transformed space implied by $\phi(\cdot)$. As usual in the context of kernel methods we start by proving the following representer theorem:

Theorem 1. *A representer theorem of the form:*

$$\underline{w}_i = \sum_j \alpha_{i,j}\phi(x_j) \tag{6.8}$$

holds true at each iteration step of the EM-procedure described by Eq.6.4-6.7.

Proof. Let us rewrite Eq.6.7 expanding the matrix product:

$$\begin{aligned} w_{i,j} &= \sum_l [\mathbf{A}^{-1}]_{i,l}[\gamma\mathbf{C} + \lambda\mathbf{B}]_{l,j} \\ &= \sum_l [\mathbf{A}^{-1}]_{i,l} \sum_m (\gamma[\phi(x_m)]_j \delta(u_m, l)+ \\ &+ \lambda[\phi(x_m)]_j \delta(u_m, 0)\delta(i,1)+ \\ &+ \lambda[\phi(x_m)]_j \delta(u_m, N_C+1)\delta(i, N_C)) \end{aligned}$$

So that:

$$\begin{aligned} \underline{w}_i = &\sum_{l,m} [\mathbf{A}^{-1}]_{i,l}(\gamma\delta(u_m, l)+ \\ &+ \lambda\delta(u_m,0)\delta_{i,1} + \lambda\delta(u_m, N_C+1)\delta_{i,N_C})[\phi(x_m)] \end{aligned}$$

Setting $\alpha_{i,j} = \sum_l [\mathbf{A}^{-1}]_{i,l}(\gamma\delta(u_m, l) + \lambda\delta(u_m, 0)\delta_{i,1} + \lambda\delta(u_m, N_C+1)\delta_{i,N_C})$ closes the proof. □

The following two corollaries complete the derivation of the EM optimization algorithm in kernel space.

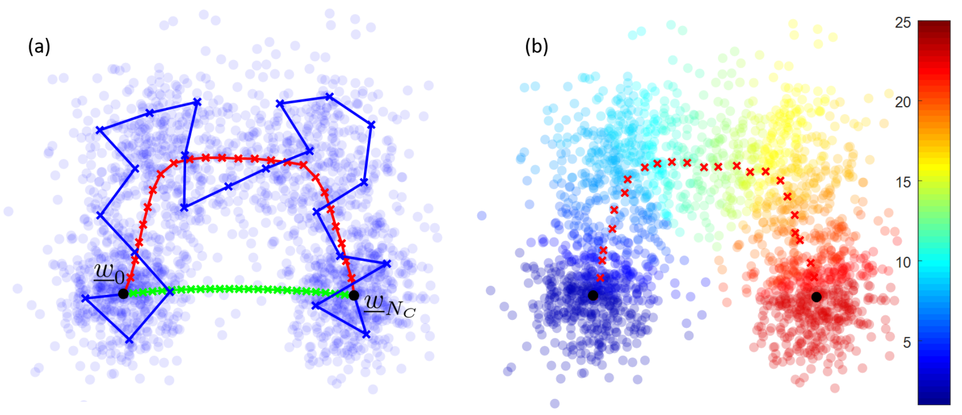

Figure 6.2: (a) Pictorial description of γ and λ trade-off: if $\gamma >> \lambda$ (blue) the path found is noisy due to an overfitting effect, for $\lambda >> \gamma$ (green) the path found approach a trivial straight line connecting the boundary points. Meaningful paths emerge for values of γ and λ in the middle of such range. (b) Typical output of the algorithm in kernel space, with latent labels represented as a color gradient.

Corollary 1.1. *The primal cost expressed by Eq.6.3 has the following dual formulation in terms of the dual variables $\alpha_{i,j}$ arranged in a vector shape $(\underline{\alpha})_{iN+j} = \alpha_{i,j}$:*

$$\Omega(\underline{\alpha}, \underline{u}, \mathbf{K}, \gamma, \lambda) = \frac{\gamma}{2}(\underline{\alpha}^T \tilde{\mathbf{A}}_{\mathbf{X}} \underline{\alpha} - 2\underline{\alpha}^T \tilde{\mathbf{K}} \underline{\delta} + \mathrm{Tr}(\mathbf{D}_{\mathbf{X}}))+$$

$$+ \frac{\lambda}{2}(\underline{\alpha}^T \tilde{\mathbf{A}}_{\mathbf{W}} \underline{\alpha} - 2\underline{\alpha}^T \tilde{\mathbf{K}} \underline{\alpha}_B + \mathrm{Tr}(\mathbf{D}_{\mathbf{W}})) \quad (6.9)$$

where $\tilde{\mathbf{A}}_{\mathbf{X}} = \mathbf{A}_{\mathbf{X}} \otimes \mathbf{K}$, $\tilde{\mathbf{A}}_{\mathbf{W}} = \mathbf{A}_{\mathbf{W}} \otimes \mathbf{K}$ and $\tilde{\mathbf{K}} = \mathbf{I}_{\mathbf{N_C}} \otimes \mathbf{K}$, $(\underline{\delta})_{iN+j} = \delta(u_i, j)$ and $\underline{\alpha}_B$ is the dual vector holding the representation of the boundaries w_0 and w_{N_C+1}.

Corollary 1.2. *The following self-consistent procedure for updating the latent labels holds true:*

$$u_i \leftarrow \arg\min_j \sum_{l,m} \alpha_{j,l}\alpha_{j,m}\mathbf{K}_{l,m} - 2\sum_l \alpha_{j,l}\mathbf{K}_{i,l} \quad (6.10)$$

$$\alpha_{i,j} \leftarrow \sum_l [\mathbf{A}^{-1}]_{i,l}(\gamma\delta(u_m, l) + \lambda\delta(u_m, 0)\delta_{i,1} + \lambda\delta(u_m, N_C + 1)\delta_{i,N_C}) \quad (6.11)$$

Proof. Eq.6.10 can be obtained by substituting Eq.6.8 into Eq.6.4. Eq.6.11 is obtained together with the representer theorem proof in the appendices and is here reported for the sake of clarity. □

Given a kernel matrix $\mathbf{K}$, a number of prototypes N_C, a value for the hyperparameters γ and λ and an initial set of labels $\underline{u}_{t_0}$, Eq.6.10-6.11 can be iterated till convergence. As shown in Fig.6.2(b), the output of the algorithm will be a final set of labels $\underline{u}_{t_f}$ from which an approximated version of the clusters prototypes in the original data space can be retrieved for example by a medoid approximation.

Empty clusters We close this section with a remark about empty-clusters. Indeed let us assume that at a given iteration t of the algorithm one of the N_C clusters has cardinality $|w_j^t|= 0$ i.e. there is no data samples assigned to it. This situation can easily arise when the number of clusters N_C is too high or when the value of the regularization parameter s is so high that some of the clusters centers are pulled away from the data. One can easily deal with such scenario in linear space (where the coordinates of $\underline{w}_j$ are readily available) simply avoiding to update the empty cluster in that iteration i.e. $\underline{w}_j^t \leftarrow \underline{w}_j^{t-1}$. However in kernel space (where the prototypes are implicitly represented with the label vector $\underline{u}$) if none of the samples are assigned to cluster $\underline{w}_j$ then one loses completely the capability of keeping track of it. For this reason we propose to deal with such situation simply removing the empty prototype $\underline{w}_j$ from the topology connecting its previous cluster center $\underline{w}_{j-1}$ to the next one $\underline{w}_{j+1}$ so that $N_C^t \leftarrow N_C^t - 1$

6.4 Approximated algorithm in kernel space

We discuss in the following how the M-step of the EM procedure minimizing Eq.6.3 can be carried out approximately without directly solving the linear system described by Eq.6.7. Two different approaches will be explored, first we will expand the recursive relation up to $O(s^2)$ terms and secondly we will present an approach based on fixed point iterations.

6.4.1 Approximation up to $O(s^2)$

Let us explicitly write the maximization step deriving $\Omega(\mathbf{W}, \underline{u}, \mathbf{X}, \gamma, \lambda)$ with respect to a given prototype $\underline{w}_j$:

$$\begin{aligned}\frac{\partial \Omega}{\partial \underline{w}_j} &= -\gamma \sum_i (\phi(\underline{x}_i) - \underline{w}_j)\delta(u_i, j) - \frac{\lambda}{2}(\underline{w}_{j+1} - \underline{w}_j) + \\ &+ \frac{\lambda}{2}(\underline{w}_j - \underline{w}_{j-1}) \\ &= -\gamma |w_j| \frac{\mathbf{C}_{j,\cdot}}{|w_j|} + (\gamma |w_j| + \lambda)\underline{w}_j - \frac{\lambda}{2}(\underline{w}_{j+1} + \underline{w}_{j-1})\end{aligned}$$

now setting $\frac{\partial\Omega}{\partial \underline{w}_j} = 0$ the following equation is derived:

$$\begin{aligned} \underline{w}_j &= \psi(\underline{w}_{j+1}, \underline{w}_{j-1}) \\ &= (1 - S_j)\frac{\mathbf{C}_{j,\cdot}}{|w_j|} + \frac{1}{2}S_j(\underline{w}_{j+1} + \underline{w}_{j-1}) \end{aligned} \tag{6.12}$$

with $S_j = \frac{s}{|w_j|+s}$. It should be clear how such formulation of the M-step has a recursive nature, as expected for a linear system of the Toeplitz type, :

$$\begin{cases} \underline{w}_{j-1} & = \psi(\underline{w}_{j-2}, \underline{w}_j) \\ \underline{w}_j & = \psi(\underline{w}_{j-1}, \underline{w}_{j+1}) \\ \underline{w}_{j+1} & = \psi(\underline{w}_j, \underline{w}_{j+2}) \end{cases}$$

Now explicitly recurring Eq.6.12 retaining just the terms up to $O(s^2)$ the following approximated M-step can be obtained:

$$\underline{w}_j \approx (1 - S_j)\frac{\mathbf{C}_{j,\cdot}}{|w_j|} + \frac{1}{2}S_j(\frac{\mathbf{C}_{j+1,\cdot}}{|w_{j+1}|} + \frac{\mathbf{C}_{j-1,\cdot}}{|w_{j-1}|}) \tag{6.13}$$

Recalling the definition of $\mathbf{C}_{j,\cdot} = \sum_i \phi(\underline{x}_i)\delta(u_i, j)$ it is trivial to observe that Eq.6.13 represents a convex combination between the centroid of cluster j and the average of the centroids of clusters $j+1$ and $j-1$. Such convex combination is ruled by the regularization parameter s so that for $s \to 0$ one recover the standard k-means M-step:

$$\underline{w}_j \approx \frac{\mathbf{C}_{j,\cdot}}{|w_j|}$$

and for $s \to \infty$ one obtains:

$$\underline{w}_j \approx \frac{1}{2}(\frac{\mathbf{C}_{j+1,\cdot}}{|w_{j+1}|} + \frac{\mathbf{C}_{j-1,\cdot}}{|w_{j-1}|})$$

The resulting EM procedure, obtained replacing Eq.6.7 with Eq.6.13 is the same as the one used for regular k-means clustering where a further smoothing step is introduced i.e.

- Update labels: $u_i \leftarrow \arg\min_j \|\phi(\underline{x}_i) - \underline{w}_j\|^2$
- Update centroids: $\underline{w}_j \leftarrow \frac{\sum_i \phi(\underline{x}_i)\delta(u_i,j)}{|w_j|}$
- Smoothing step: $\underline{w}_j \leftarrow (1 - S_j)w_j + \frac{S_j}{2}(\underline{w}_{j+1} + \underline{w}_{j-1})$

We now prove that the expectation step of the procedure as described by Eq.6.4 can be carried out without the explicit notion of $\phi(x)$:

$$
\begin{aligned}
u_i \leftarrow & \arg\min_j \|\phi(\underline{x}_i) - \underline{w}_j\|^2 \\
& = \arg\min_j 2(S_j - 1)\langle \phi(\underline{x}_i), \frac{C_j}{|w_j|}\rangle - S_j \langle \phi(x_i), \frac{C_{j-1}}{|w_{j-1}|} + \frac{C_{j+1}}{|w_{j+1}|}\rangle + \\
& + (1 - S_j)^2 \langle \frac{C_j}{|w_j|}, \frac{C_j}{|w_j|}\rangle + \frac{S_j^2}{4}\langle \frac{C_{j-1}}{|w_{j-1}|} + \frac{C_{j+1}}{|w_{j+1}|}, \frac{C_{j-1}}{|w_{j-1}|} + \frac{C_{j+1}}{|w_{j+1}|}\rangle + \\
& - (1 - S_j)S_j \langle \frac{C_j}{|w_j|}, \frac{C_{j-1}}{|w_{j-1}|} + \frac{C_{j+1}}{|w_{j+1}|}\rangle
\end{aligned}
\tag{6.14}
$$

observing that $\langle \frac{C_i}{|w_i|}, \frac{C_j}{|w_j|}\rangle = \frac{1}{|w_i||w_j|}\sum_{l,m} K_{l,m}\delta(u_l, i)\delta(u_m, j)$ prove that Eq.6.14 can be carried out with the only notion of the kernel matrix $\mathbf{K}$.

6.4.2 Approximation by fixed point iterations

We present here an alternative approach to the problem of approximating the M-step of the algorithm in kernel space. In the the previous section we showed indeed how an approximated M-step can be obtained by stopping the recursion of Eq.6.12 at $O(s^2)$. Whereas we propose here to disentangle such recursive system evaluating the left-hand side at iteration $T + 1$ and the right-hand side at iteration T:

$$
\begin{cases}
\underline{w}_{j-1}^{T+1} & = \psi(\underline{w}_{j-2}^T, \underline{w}_j^T) \\
\underline{w}_{j,T+1} & = \psi(\underline{w}_{j-1}^T, \underline{w}_{j+1}^T) \\
\underline{w}_{j+1}^{T+1} & = \psi(\underline{w}_j^T, \underline{w}_{j+2}^T)
\end{cases}
\tag{6.15}
$$

i.e. we treat the iterations of the proposed algorithm as fixed point iterations which will converge to the solution starting from the initial guess $\underline{w}_j^0$.

We now postulate that at a given iteration T of the algorithm we can approximately represent the prototype $\underline{w}_j^T$ as a weighted linear combination of the samples :

$$
\underline{w}_j^T \approx \widehat{\underline{w}}_j^T = \sum_{i=0}^{N-1} \widehat{\alpha}_i^T \phi(x_i)
\tag{6.16}
$$

Where the hat over the coefficients indicate that they are not the one obtained analytically in the demonstration of the representer theorem for the exact algorithm. The following approximate M-step is derived:

$$
\underline{w}_j^{T+1} = \psi(\widehat{\underline{w}}_{j-1}^T, \widehat{\underline{w}}_{j+1}^T) = (1 - S_j)\frac{C_j}{|w_j|} + \frac{S_j}{2}\{\widehat{\underline{w}}_{j-1}^T + \widehat{\underline{w}}_{j+1}^T\}
\tag{6.17}
$$

As usual one can show that such step can be carried out implicitly in kernel space

$$\begin{aligned}
u_i^T &\leftarrow \arg\min_j \|\phi(x_i) - w_j^T\|^2 \\
&= \arg\min_j 2(S_j - 1)\langle\phi(x_i), \frac{C_j}{|w_j|}\rangle - S_j\langle\phi(x_i), \underline{\widehat{w}}_{j-1}^T + \underline{\widehat{w}}_{j+1}^T\rangle + \\
&+ (1 - S_j)^2\langle\frac{C_j}{|w_j|}, \frac{C_j}{|w_j|}\rangle + \frac{S_j^2}{4}\langle\underline{\widehat{w}}_{j-1}^T + \underline{\widehat{w}}_{j+1}^T, \underline{\widehat{w}}_{j-1}^T + \underline{\widehat{w}}_{j+1}^T\rangle + \\
&- (1 - S_j)S_j\langle\frac{C_j}{|w_j|}, \underline{\widehat{w}}_{j-1}^T + \underline{\widehat{w}}_{j+1}^T\rangle
\end{aligned} \tag{6.18}$$

We can therefore control the accuracy of the EM-like procedure choosing different representations $\underline{\widehat{w}}_j$ i.e. different approximated weight vectors $\widehat{\alpha}_{\cdot,j}$. In the following paragraphs we list some possibilities detailing their computational complexity. One should appreciate how the approach presented here has a different nature with respect to the one presented in the previous section. Indeed the approximated algorithm derived here is formulated in terms of the dual variables $\widehat{\alpha}_{\cdot,j}$.

Centroid representation: one can set $\underline{w}_j^T \approx \underline{\widehat{w}}_j^T = \frac{C_j^T}{|w_j|}$, this is equivalent to the following definition for the approximated weight vector $\widehat{\alpha}$:

$$\widehat{\alpha}_{i,j} = \sum_{j=0}^{N_c-1} \frac{1}{|w_j|}\delta(y_i, j) \tag{6.19}$$

A graphical representation of the derived M-step is shown in Fig.6.3. Such approximation will be valid in the limit of $s \to 0$ i.e. small values for the regularization parameter so that the cluster prototypes are not expected to move too much from the standard kernel k-means solution. Indeed substituting $\underline{\widehat{w}}_j^T = \frac{C_j^T}{|w_j|}$ in Eq.6.18 one can easily verify that the $O(s^2)$ approximation described by Eq.6.14 is obtained. Therefore such simple representation does not introduce any computational overhead but does not improve the accuracy of the algorithm neither.

Medoid representation: Another possibility is to approximate $\underline{w}_j^T$ simply with the closest sample in kernel space i.e. setting the weight vector $\widehat{\alpha}$ to be:

$$\widehat{\alpha}_{i,j} = \begin{cases} 1 & \text{if } \|\phi(\underline{x}_i) - \underline{w}_j^T\|^2 < \|\phi(\underline{x}_l) - \underline{w}_j^T\|^2, \forall j \in [0, N_C - 1], \forall l \in [0, N - 1] \\ 0 & \text{otherwise} \end{cases} \tag{6.20}$$

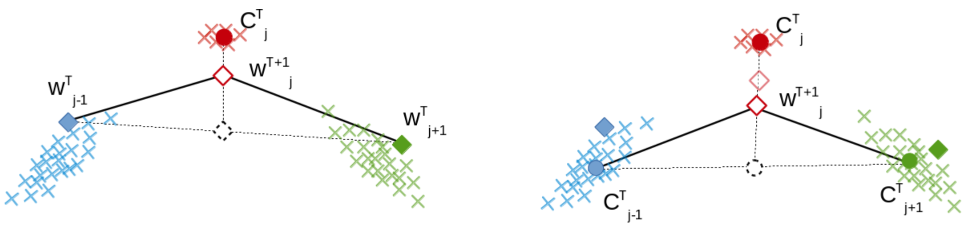

Figure 6.3: On the left a graphical rapresentation of the exact M-step is shown. On the right instead the approximated M-step obtained by stopping the recursion up to $O(s^2)$ terms or equivalently approximating $\underline{w}_j^T \approx \widehat{\underline{w}}_j^T = \frac{C_j^T}{|w_j|}$ is shown.

Such medoid representation introduce an additional $O(N_C N)$ computational cost but evaluating it in term of accuracy gain is not trivial. Indeed the approximation will be accurate provided a sufficient dense dataset i.e. provided that prototypes are not too far form data samples.

Min squared weighted representation: The best solution to the representation problem in kernel space should be learning the weight vector $\widehat{\alpha}_{\cdot,j}$ minimizing the square distance between the approximate representations and the actual prototypes i.e.:

$$\begin{aligned}\widehat{\alpha}_{\cdot,j}^T &= \arg\min_{\widehat{\alpha}_{\cdot,j}^T} \frac{1}{2} \| \sum_{i=0}^{N-1} \sum_{j=0}^{N_C-1} (\underline{w}_j^T - \alpha_{i,j}^T \phi(\underline{x}_i)) \delta(u_i^T, j) \|^2 \\ &= \arg\min_{\widehat{\alpha}_{\cdot,j}^T} \frac{1}{2} \| \sum_{i=0}^{N-1} \sum_{j=0}^{N_C-1} ((1-S_j)\frac{C_j^{T-1}}{|w_j|} + \frac{S_j}{2}(\widehat{\underline{w}}_{j-1}^{T-1} + \widehat{\underline{w}}_{j+1}^{T-1}) + \\ &- \alpha_i^T \phi(\underline{x}_i)) \delta(u_i^T, j) \|^2 \end{aligned} \tag{6.21}$$

Deriving with respect to a generic $\widehat{\alpha}_{l,j}$ and setting the derivative to 0 we have:

$$\begin{aligned}\sum_{i=0}^{N-1} \delta(u_i^T, j) \widehat{\alpha}_{i,j}^T \langle \phi(\underline{x}_l), \phi(\underline{x}_i) \rangle = |\underline{w}_j| \{ (1-S_j) \langle \phi(\underline{x}_l), \frac{C_j^{T-1}}{|w_j|} \rangle + \\ + \frac{S_j}{2} \langle \phi(\underline{x}_l), \widehat{\underline{w}}_{j-1}^{T-1} + \widehat{\underline{w}}_{j+1}^{T-1} \rangle \}, \forall j \in [0, N_C - 1] \end{aligned} \tag{6.22}$$

Therefore, for each prototype $\underline{w}_j$ we would need to solve a linear system in $|w_j|$ unknowns. This is the most accurate representation that we can get in kernel space however (assuming a constant cluster cardinality of $\frac{N}{N_c}$) it requires $O(\frac{N^3}{N_C^2})$ operations. It is important to notice that even though this solution is theoretically valuable, in practice does not offer any computational advantages with respect to the exact algorithm.

A priori sparse min squared weighted representation: In order to reduce the computational complexity of the minimum squared weighted representation introduced in the previous paragraph, one can restrict $\underline{\widehat{w}}_j$ to the subspace spanned by a subset $\mathbf{L} \subset \mathbf{X}$ of landmarks i.e.:

$$\underline{w}_j^T \approx \underline{\widehat{w}}_j^T = \sum_{i:\underline{x}_i \in \mathbf{L}} \widehat{\alpha}_{i,j}^T \phi(\underline{x}_i) \tag{6.23}$$

The set of landmarks can be selected once at the beginning of the algorithm or it can be selected by a given rational at each iteration e.g. so that the number of landmarks per cluster is constant. One can easily prove that the computational overhead for such a representation is reduced to $O(\frac{|L|^3}{N_C^2})$.

6.5 Manifold selection: filtering data

In chapter 3 we introduced the concept of Principal Path in Data Space as a local version of Principal Curves (PCs). Where local implies the fact that the path is not required to *pass through* the entire set of data. For example in a constellation-like dataset, as the one depicted in Fig.6.4(a), the desired path lies on a local manifold that does not involve the whole dataset. However the introduced algorithm, being derived as a regularized version of kernel k-means, is intrinsically global. Thus, to focus on submanifolds connecting $\underline{w}_0$ to $\underline{w}_{N_C+1}$ we discuss here a possible pre-filtering scheme on the dataset to be applied before the actual run of the algorithm. Note that, in case a global manifold is desired this filtering procedure can be safely turned off.

The filtering procedure can be summarized by the following points:

(a) Select boundary points i.e. $\underline{w}_0$ and $\underline{w}_{N_C+1}$.

(b) Extract a set of N_f medoids $\mathbf{M} = \{\underline{m}_i\} \subset \mathbf{X}$ with kernel k-means++ algorithm [39].

(c) Build a penalized k-nearest-neighbor graph represented by a penalized distance matrix among $\mathbf{M}$ with penalty factor p:

$$d_p(\underline{m}_i, \underline{m}_j) = \begin{cases} d(\underline{m}_i, \underline{m}_j) & \underline{m}_i \in \mathrm{nn}_k(\underline{m}_j) \\ pd(\underline{m}_i, \underline{m}_j) & \text{otherwise} \end{cases}$$

.

(d) Run Dijkstra algorithm over the penalized distance matrix $d_p(\underline{m}_i, \underline{m}_j)$ in order to find the shortest path connecting $\underline{w}_0$ to $\underline{w}_{N_C+1}$. Such path will be represented as an ordered subset of medoids $\mathbf{S} \subset \mathbf{M}$.

(e) Remove from $\mathbf{M}$ the points that are too close to the path's medoids $\mathbf{S}$ with respect to an arbitrary threshold T i.e.

$$\forall \underline{m}_i \in (\mathbf{M} \setminus \mathbf{S}) : \quad \mathbf{M} \leftarrow \mathbf{M} \setminus \underline{m}_i \quad \text{if} \quad d(\underline{m}_i, \mathbf{S}) < T$$

.

(f) Label the samples according to the closest medoid. The data to be filtered out will be the one associated with medoids in the set $\mathbf{M} \setminus \mathbf{S}$.

The different filtering phases are illustrated on a synthetic constellation-like dataset in Fig.6.4(a)-(f) and Fig.6.5(a)-(f) for different choices of the boundary conditions.

6.6 Model selection via Bayesian evidence maximization

In the following section we discuss how γ and λ can be selected through the Bayesian framework of Maximal Evidence. At first we show how the primal problem introduced by Eq.6.3 can be framed in the context of Bayesian inference as a posterior maximization problem. In doing so we also reproduce the steps originally introduced by MacKay [24] in order to derive an analytical expression for the Bayesian evidence in our problem of interest.

We then give a proper Bayesian interpretation for both γ and λ parameters, as it will be clear this will give us useful insights about the very nature of the two hyperparameters allowing, as it will be discussed in the experimental section to heuristically set γ thus reducing the dimensionality of the parameter space.

We will then discuss the needed approximations to numerically evaluate the analytical evidence expression in order to define a useful protocol for model selection both in linear and kernel space. We stress here the fact that, even though the Maximal Evidence framework was successfully applied to several supervised learning problems, its application in the unsupervised domain is unexplored and may represent a major contribution to the field.

6.6.1 From cost minimization to posterior maximization

As a starting point we show how the cost function minimization process can be regarded as a Bayesian posterior maximization problem for the model described by the set of prototypes $\mathbf{W}$. We can think to Ω as an energy function to be minimized, thus taking inspiration from classical statistical mechanics we set the energy to be proportional to the negative logarithm of

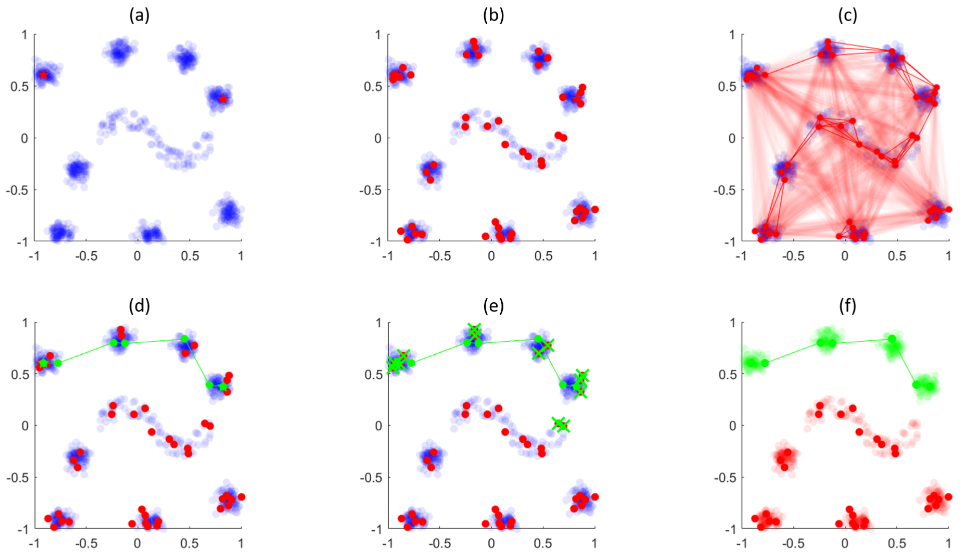

Figure 6.4: (a)-(f) Filtering phases as described in section 6.5 in order to focus on the sub-manifold implied by the choice of boundary conditions.

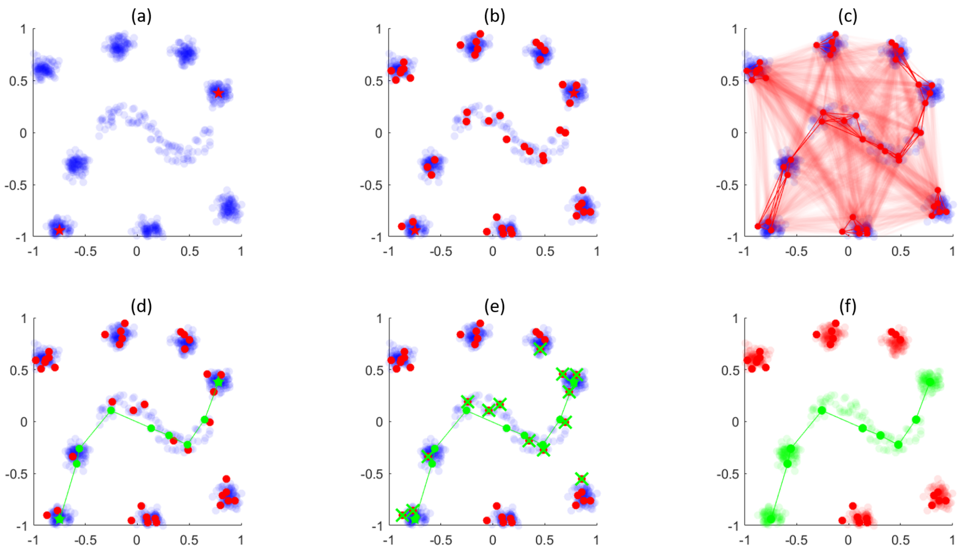

Figure 6.5: Same filtering phase with a different choice of boundary conditions. This shows how the filtering scheme is able to select different local sub-manifolds in a constellation-like dataset.

the posterior probability $P(\mathbf{W}|\mathbf{X},\gamma,\lambda)$ i.e.

$$\begin{aligned} P(\mathbf{W}|\mathbf{X},\gamma,\lambda) &= P(\mathbf{X}|\mathbf{W},\gamma,\lambda)P(\mathbf{W}|\gamma,\lambda)\frac{1}{P(\mathbf{X}|\gamma,\lambda)} \\ &= \frac{e^{-\Omega(\mathbf{W},\mathbf{X},\gamma,\lambda)}}{Z(\gamma,\lambda)} \\ &= \frac{e^{-\gamma\Omega_X(\mathbf{W},\mathbf{X})}}{Z_X(\gamma)}\frac{e^{-\lambda\Omega_W(\mathbf{W})}}{Z_W(\lambda)}\frac{Z_X(\gamma)Z_W(\lambda)}{Z(\gamma,\lambda)} \end{aligned} \tag{6.24}$$

where Z, Z_X and Z_W are the partition functions obtained integrating out all the degrees of freedom respectively of the global cost, of the k-means cost and of the regularizer cost:

$$Z(\gamma,\lambda) = \int e^{-\Omega(\mathbf{W},\mathbf{X},\gamma,\lambda)}dW$$

$$Z_D(\gamma) = \int e^{-\gamma\Omega_X(\mathbf{W},\mathbf{X})}dW$$

$$Z_W(\lambda) = \int e^{-\lambda\Omega_W(\mathbf{W})}dW$$

Comparing the first and the last lines of Eq.6.24 we can obtain the following definition for the likelihood of the data:

$$L(\gamma,\mathbf{W}) = P(\mathbf{X}|\mathbf{W},\gamma,\lambda) = \frac{e^{-\gamma\Omega_X(\mathbf{W},\mathbf{X})}}{Z_X(\gamma)} \tag{6.25}$$

,

for the prior probability of the model:

$$P(\mathbf{W},\lambda) = \frac{e^{-\lambda\Omega_W(\mathbf{W})}}{Z_W(\lambda)} \tag{6.26}$$

,

and for the evidence:

$$E(\gamma,\lambda) = P(\mathbf{X}|\gamma,\lambda) = \frac{Z(\gamma,\lambda)}{Z_X(\gamma)Z_W(\lambda)} \tag{6.27}$$

.

Hereafter we will refer to the maximum posterior solution as $\mathbf{W}_{\text{MP}}$:

$$\mathbf{W}_{\text{MP}} = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{X},\gamma,\lambda) = \arg\min_{\mathbf{W}} \Omega(\mathbf{W},\mathbf{X},\gamma,\lambda)$$

We have now introduced all the quantities needed to infer both the model, i.e. set of prototypes $\mathbf{W}$, and the hypothesis i.e. set of parameters (γ,λ) starting from the training data $\mathbf{X}$. Thus the two level Bayesian inference procedure that we aim at can be summarized as follow:

- 1st Level: starting from a given hypothesis (γ, λ) infer the best model $\mathbf{W}_{MP}$ through a maximum posterior criterion (i.e. minimum cost) using the algorithm introduced in the previous sub-section.
- 2nd Level: Infer the best parameters (γ, λ) with a maximum posterior criteria on the hypothesis set. Assuming a flat prior probability (i.e. stopping the inference at this second level) such maximum posterior criteria is equivalent to a maximum evidence criteria as shown by:

$$\begin{aligned}(\gamma, \lambda)_{\mathrm{ME}} &= \max_{\gamma,\lambda} P(\gamma, \lambda|\mathbf{X}) \\ &= \max_{\gamma,\lambda} \frac{P(\mathbf{X}|\gamma, \lambda)P(\gamma, \lambda)}{P(\mathbf{X})} \\ &= \max_{\gamma,\lambda} E(\gamma, \lambda)\end{aligned}$$

6.6.2 Bayesian interpretation of γ and λ

Before going on with the numerical evaluation of the evidence just introduced we take advantage of the Bayesian reformulation of the problem to get insights about the nature of the algorithm hyperparameters.

Bayesian interpretation of γ

Observing Eq.6.25 the interpretation of γ is straightforward. Indeed :

$$\begin{aligned}P(\mathbf{X}|\mathbf{W}, \gamma, \lambda) &= \prod_{i=1}^{N} P(\underline{x}_i|\mathbf{W}, \gamma, \lambda) \\ &= \prod_{i=1}^{N} \exp(-\sum_j \gamma\|\underline{x}_i - \underline{w}_j\|^2\delta(u_i, j))\end{aligned} \tag{6.28}$$

i.e. Assuming that each cluster j is generated sampling a gaussian distribution centered around w_j, then γ represents the variance of such distribution. It is worth observing that this result was already implicitly derived in chapter **??** as one of the motivations behind the choice of kernel k-means as a valuable clustering algorithm for MD trajectories. There indeed we derived the kernel k-means cost function starting from a maximum likelihood principle, assuming a likelihood in the form of Eq.6.25 due to the natural smoothness of the energy surface from which MD data are sampled.

Bayesian interpretation of λ

The interpretation of λ starting from the Bayesian prior definition given in Eq.6.26 is slightly more involved than the one for γ. Let us start rewriting

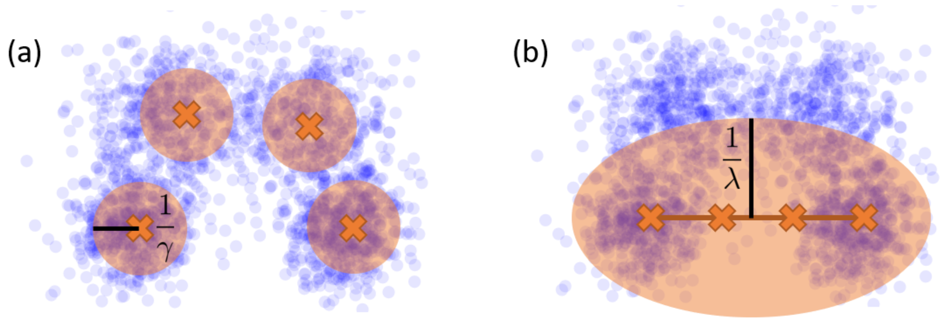

Figure 6.6: (a) Pictorial representation of the Bayesian interpretation of γ as the variance of a gaussian mixture model with gaussians centered around the cluster prototypes. (b) Pictorial representation of the Bayesian interpretation of λ as the variance of a gaussian prior centered around the straight line connecting the starting and the ending cluster prototypes.

the regularized cost function completing the square:

$$\begin{aligned}\Omega_W(\mathbf{W}) &= \mathrm{Tr}(\mathbf{W}^T\mathbf{A_W}\mathbf{W} - \mathbf{B}^T\mathbf{W} + \mathbf{D_W})\\ &= \mathrm{Tr}((\mathbf{W} - \frac{1}{2}\mathbf{A_W}^{-1}\mathbf{B})^T\mathbf{A_W}(\mathbf{W} - \frac{1}{2}\mathbf{A_W}^{-1}\mathbf{B}) +\\ &+ \mathbf{D_W} - \frac{1}{4}\mathbf{B}^T\mathbf{A_W}^{-1}\mathbf{B})\end{aligned}$$

Substituting such formulation of the regularized cost into Eq.6.26 we obtain:

$$P(\mathbf{W},\lambda) \propto e^{-\lambda\,\mathrm{Tr}((\mathbf{W}-\mathbf{W}^{\#})^T\mathbf{A_W}(\mathbf{W}-\mathbf{W}^{\#}))} \tag{6.29}$$

where we have defined the matrix $W^{\#}$ as:

$$\mathbf{W}^{\#} = \frac{1}{2}\mathbf{A_W}^{-1}\mathbf{B}$$

$$\begin{bmatrix} \dots & \underline{w}_1^{\#} & \dots \\ & \vdots & \\ \dots & \underline{w}_i^{\#} & \dots \\ & \vdots & \\ \dots & \underline{w}_{N_C}^{\#} & \dots \end{bmatrix} = \frac{1}{2}\mathbf{A_W}^{-1} \begin{bmatrix} \dots & \underline{w}_0 & \dots \\ \dots & 0 & \dots \\ & \vdots & \\ \dots & 0 & \dots \\ \dots & \underline{w}_{N_C+1} & \dots \end{bmatrix}$$

Eq.6.29 tells us that λ represents the variance of a gaussian prior centered around $W^{\#}$. To better understand the meaning of such result we take advantage of the fact that the analytical inversion of tridiagonal matrices in the form of A_W is known [69], indeed we have:

$$\begin{aligned}(\frac{1}{2}\mathbf{A_W}^{-1})_{1,j} &= \frac{NC+1-j}{NC+1} \\ (\frac{1}{2}\mathbf{A_W}^{-1})_{j,N_C} &= \frac{j}{NC+1}\end{aligned} \tag{6.30}$$

it follows that:

$$\underline{w}_i^{\#} = \frac{N_C+1-i}{N_C+1}\underline{w}_0 + \frac{i}{N_C+1}\underline{w}_{N_C+1} \tag{6.31}$$

i.e. the regularizer cost function describes a gaussian prior with variance λ centered around the straight line connecting the boundaries $\underline{w}_0$ and $\underline{w}_{N_C+1}$.

For the sake of clearness a pictorial description of both γ and λ bayesian interpretation is given in Fig.6.6.

6.6.3 Numerical evidence evaluation in Linear Space

Now that the learning problem is well framed in the framework of Bayesian inference we discuss here how it is possible to practically evaluate the (log)-evidence introducing some approximations since a straightforward application of the analytical derivation is unfeasible.

For the sake of clarity we write here the logarithmic equation for the evidence:

$$\log E(\gamma, \lambda) = \log Z(\gamma, \lambda) - \log Z_X(\gamma) - \log Z_W(\lambda) \tag{6.32}$$

it is thus clear that one needs to evaluate the three terms $Z(\gamma, \lambda)$, $Z_X(\gamma)$, $Z_W(\lambda)$. In the following we discuss the evaluation of all the three terms.

partition function Z_W among the three partition functions $Z_W(\lambda)$ is the simplest to evaluate, indeed being Ω_W a quadratic form with respect to $\mathbf{W}$, such integral can be evaluated analytically as per:

$$Z_W(\lambda) = \left(\frac{(2\pi)^{N_C}}{\det(\lambda \mathbf{A_W})}\right)^{\frac{d}{2}} e^{\frac{\lambda}{8}\operatorname{Tr}(\mathbf{B}^T\mathbf{A_W}^{-1}\mathbf{B})} e^{-\lambda \operatorname{Tr}(\mathbf{D_W})}$$

taking the logarithm and retaining only the terms depending on λ we have:

$$\log Z_W = -\frac{dN_C}{2}\log\lambda + \frac{\lambda}{8}\operatorname{Tr}(\mathbf{B}^T\mathbf{A_W}^{-1}\mathbf{B}) - \lambda \operatorname{Tr}(\mathbf{D_W})$$

k-means partition function Z_X The evaluation of the k-means partition function is not as straightforward as the previous one since the k-means cost function Ω_X is not quadratic. We propose here a quadratic approximation around the unregularized k-means maximum posterior solution $\mathbf{W}_{UMP}$ obtained by minimizing $\Omega_X(\mathbf{W}, \mathbf{X})$:

$$\Omega_X(\mathbf{W}, \mathbf{X}) \approx \Omega_X(\mathbf{W}_{UMP}, \mathbf{X}) + \mathrm{Tr}(\frac{1}{2}(\mathbf{W} - \mathbf{W}_{UMP})^T \mathbf{A}_\mathbf{X}|_{\mathbf{W}_{UMP}} (\mathbf{W} - \mathbf{W}_{UMP}))$$

.

At this point the evaluation of Z_X is reduced to a standard gaussian integral:

$$Z_X(\gamma) \approx \left(\frac{(2\pi)^{N_C}}{\det(\gamma \mathbf{A}_\mathbf{X}|_{\mathbf{W}_{UMP}})} \right)^{\frac{d}{2}} e^{-\gamma \Omega_X(\mathbf{W}_{UMP}, \mathbf{X})}$$

taking the logarithm and retaining only the terms depending on γ and on $\mathbf{W}_{UMP}$ we have:

$$\log Z_X(\gamma) = -\frac{dN_C}{2} \log(\gamma) - \frac{d}{2} \log\det \mathbf{A}_\mathbf{X}(\mathbf{W}_{UMP}) - \gamma \Omega_X(\mathbf{W}_{UMP}, \mathbf{X})$$

Regularized k-means partition function Z The last partition function that need to be evaluated in this context is the global one, including both the contribution from the standard k-means and the one coming from the regularization term. As for Ω_X we are dealing with a non-convex cost function Ω, therefore we proceed with the same kind of quadratic approximation. In this case however the expansion will be centered around the maximum posterior solution obtained minimizing the regularized cost function i.e. $\mathbf{W}_{MP}$ so that:

$$Z(\gamma, \lambda) \approx \left(\frac{(2\pi)^{N_C}}{\det(\gamma \mathbf{A}_\mathbf{X} + \lambda \mathbf{A}_\mathbf{W})|_{\mathbf{W}_{MP}}} \right)^{\frac{d}{2}} e^{-\Omega(\mathbf{W}_{MP}, \mathbf{X}, \gamma, \lambda)} \tag{6.33}$$

taking the logarithm and retaining only the terms depending on γ, λ and $\mathbf{W}_{MP}$ we have:

$$\log Z(\gamma, \lambda) \approx -\frac{d}{2} \log\det(\gamma \mathbf{A}_\mathbf{X} + \lambda \mathbf{A}_\mathbf{W})|_{\mathbf{W}_{MP}} - \gamma \Omega_X(\mathbf{W}_{MP}, \mathbf{X}) - \lambda \Omega_W(\mathbf{W}_{MP})$$

Approximate log evidence Substituting the partial results obtained above in Eq.6.32 we have:

$$\begin{aligned}\log E(\gamma,\lambda) \approx -\frac{d}{2}\log\det(\gamma\mathbf{A}_{\mathbf{X}}+\lambda\mathbf{A}_{\mathbf{W}})|_{\mathbf{W}_{MP}}+\\ -\gamma\Omega_X(\mathbf{W}_{MP},\mathbf{X})-\lambda\Omega_W(\mathbf{W}_{MP})+\frac{dN_C}{2}\log(\gamma)+\\ +\frac{d}{2}\log\det\mathbf{A}_{\mathbf{X}}|_{\mathbf{W}_{UMP}}+\gamma\Omega_X(\mathbf{W}_{UMP},\mathbf{X})+\frac{dN_C}{2}\log\lambda+\\ -\lambda\frac{1}{8}\operatorname{Tr}(\mathbf{B}^T{\mathbf{A}_{\mathbf{W}}}^{-1}\mathbf{B}+\lambda\mathbf{D}_{\mathbf{W}})\end{aligned} \quad (6.34)$$

It is worth noting that this final formulation depends on both γ, λ independently and cannot be rewritten in terms of the single regularization parameter $s = \frac{\lambda}{\gamma}$. This proves that in order to properly address the model selection with a Bayesian framework the max-search for the evidence function has to be performed on the bidimensional space $\gamma \times \lambda$. This is an intrinsic, undesired property of the Bayesian framework (which can be found also in the simplest case of mean inference [70]). It is straightforward to demonstrate that on the line described by $\frac{\lambda}{\gamma} = \text{const}$ all the hypothesis are equivalent (i.e. the solution to the 1st level inference is unique) however the evidence is not flat on such line. In other words, even though the original cost function is scale invariant, there is a preferred scale that one has to choose in order to properly select the hypothesis through a max evidence framework. Eq.6.34 was here derived assuming an explicit knowledge of the transformed input space $\phi(\cdot) : \mathbb{R}^d \to \mathbb{R}^{d'}$, indeed one needs to know the dimensionality d' in order to evaluate it. The same result however can be derived in kernel space, indeed as we are going to demonstrate in the following section Eq.6.34 holds true replacing the dimensionality of the input space d' with an estimate of the kernel matrix rank r.

6.6.4 Numerical evidence evaluation in Kernel Space

Starting from the dual formulation of the problem given by Eq.6.9 we now discuss how it is possible to evaluate the Bayesian evidence in kernel space, without an explicit notion of the transformation $\phi(\cdot)$. We will follow closely the procedure carried out in the previous section, so as a first step one needs to evaluate the following three partition functions:

$$Z(\gamma,\lambda) = \int e^{-\Omega(\underline{\alpha},\mathbf{X},\gamma,\lambda)}d\underline{\alpha}$$

$$Z_D(\gamma) = \int e^{-\gamma\Omega_X(\underline{\alpha},\mathbf{X})}d\underline{\alpha}$$

$$Z_W(\lambda) = \int e^{-\lambda\Omega_W(\underline{\alpha})}d\underline{\alpha}$$

where we replaced the integration over the phase space spanned by $\mathbf{W}$ with an integration of the space spanned by the dual variables $\underline{\alpha}$.

As for the primal derivation, also here the integral needed to evaluate $Z_W(\lambda)$ is a known gaussian integral:

$$Z_W(\lambda) = \left(\frac{(2\pi)^{NN_C}}{\det(\lambda \tilde{\mathbf{A}}_{\mathbf{W}})} \right)^{\frac{1}{2}} e^{\frac{\lambda}{8}(\tilde{\mathbf{K}}\underline{\alpha}_B)^T \tilde{\mathbf{A}}_{\mathbf{W}}^{-1}(\tilde{\mathbf{K}}\underline{\alpha}_B)} e^{-\lambda \operatorname{Tr}(\mathbf{D}_{\mathbf{W}})}$$

The exact evaluation of $Z_X(\gamma)$ and $Z(\gamma, \lambda)$ instead is not feasible due to the non-convexity of Ω_X and Ω. We therefore proceed with a quadratic approximation around the unregularized maximum posterior solution $\underline{\alpha}_{UMP}$ and the maximum posterior solution $\underline{\alpha}_{MP}$ respectively:

$$Z_X(\gamma) \approx \left(\frac{(2\pi)^{NN_C}}{\det(\gamma \tilde{\mathbf{A}}_{\mathbf{X}}|_{\underline{\alpha}_{UMP}})} \right)^{\frac{1}{2}} e^{-\gamma \Omega_X(\underline{\alpha}_{UMP}, \mathbf{X})}$$

$$Z(\gamma, \lambda) \approx \left(\frac{(2\pi)^{NN_C}}{\det(\gamma \tilde{\mathbf{A}}_{\mathbf{X}} + \lambda \tilde{\mathbf{A}}_{\mathbf{W}})|_{\underline{\alpha}_{MP}}} \right)^{\frac{1}{2}} e^{-\Omega(\underline{\alpha}_{MP}, \mathbf{X}, \gamma, \lambda)}$$

Now taking the logarithm and neglecting all the terms which do not depend on γ and λ one derives the following equation for the approximated evidence in kernel space:

$$\begin{aligned} \log E(\gamma, \lambda) \approx & -\frac{1}{2} \log\det(\gamma \tilde{\mathbf{A}}_{\mathbf{X}} + \lambda \tilde{\mathbf{A}}_{\mathbf{W}})|_{\underline{\alpha}_{MP}} + \\ & - \gamma \Omega_X(\underline{\alpha}_{MP}, \mathbf{X}) - \lambda \Omega_W(\underline{\alpha}_{MP}) + \\ & + \frac{1}{2} \log\det \gamma \tilde{\mathbf{A}}_{\mathbf{X}}|_{\underline{\alpha}_{UMP}} + \gamma \Omega_X(\underline{\alpha}_{UMP}, \mathbf{X}) + \\ & + \frac{1}{2} \log\det \lambda \tilde{\mathbf{A}}_{\mathbf{W}} - \frac{\lambda}{8}(\tilde{\mathbf{K}}\underline{\alpha}_B)^T \tilde{\mathbf{A}}_{\mathbf{W}}^{-1}(\tilde{\mathbf{K}}\underline{\alpha}_B) + \lambda \operatorname{Tr} \mathbf{D}_{\mathbf{W}} \end{aligned} \tag{6.35}$$

It is worth noting that the condition of $\tilde{\mathbf{A}}_{\mathbf{W}}$ and $\tilde{\mathbf{A}}_{\mathbf{D}}$ matrices strongly depend on the condition of $\mathbf{K}$. Since we cannot assume $\mathbf{K}$ to be non-singular in general we have to discuss how Eq.6.35 can be evaluated when this is not the case. In particular we will discuss in the following how to correctly evaluate the terms where a determinant computation is needed.

We start observing that:

$$\gamma \tilde{\mathbf{A}}_{\mathbf{X}} + \lambda \tilde{\mathbf{A}}_{\mathbf{W}} = (\gamma \mathbf{A}_{\mathbf{X}} + \lambda \mathbf{A}_{\mathbf{W}}) \otimes \mathbf{K}$$

for the associative property of the Kronecker product.

In order to deal with singular **K** matrices in the following we will replace the determinant operator with the pseudo-determinant defined as:

$$\begin{aligned}\text{pdet}(\bullet) &= \lim_{\epsilon\to 0} \frac{\det(\bullet + \epsilon\mathbf{I})}{\epsilon^{\text{sz}(\bullet)-\text{rk}(\bullet)}} \\ &= \lim_{\epsilon\to 0} \frac{\prod_i(\mu_i^\bullet + \epsilon)}{\epsilon^{\text{sz}(\bullet)-\text{rk}(\bullet)}} \\ &= \frac{\cancel{\epsilon^{\text{sz}(\bullet)-\text{rk}(\bullet)}} \prod_{\mu_i^\bullet>0} \mu_i^\bullet}{\cancel{\epsilon^{\text{sz}(\bullet)-\text{rk}(\bullet)}}}\end{aligned}$$

It is a known result [71] that the eigenvalues of $\mathbf{A} \otimes \mathbf{K}$ are given by all possible products $\{\mu_i^{\mathbf{A}}\mu_j^{\mathbf{K}}\}$ of the **A** and **K** eigenvalues. Therefore, being **A** a full-rank matrix and being **K** a positive semi-definite matrix of rank $\text{rk}(\mathbf{K}) = |\mu_i^K > 0| = r$ we have:

$$\begin{aligned}\text{pdet}(\mathbf{A}\otimes\mathbf{K}) &= \prod_{\mu_i^{\mathbf{A}}\mu_j^{\mathbf{K}}>0} \mu_i^{\mathbf{A}}\mu_j^{\mathbf{K}} \\ &= \prod_{\mu_j^{\mathbf{K}}>0} (\mu_1^{\mathbf{A}}\mu_j^{\mathbf{K}} \dots \mu_{\text{sz}(\mathbf{A})}^{\mathbf{A}}\mu_j^{\mathbf{K}}) \\ &= \prod_{\mu_j^{\mathbf{K}}>0} (\mu_1^{\mathbf{A}} \dots \mu_{\text{sz}(\mathbf{A})}^{\mathbf{A}})(\mu_j^{\mathbf{K}})^{\text{sz}(\mathbf{A})} \\ &= \det(\mathbf{A})^r \,\text{pdet}(\mathbf{K})^{\text{sz}(\mathbf{A})}\end{aligned}$$

The evaluation of the determinant terms appearing in Eq.6.35 is now straightforward:

$$\begin{aligned}\frac{1}{2}\log\text{pdet}(\gamma\tilde{\mathbf{A}}_{\mathbf{X}}|_{\underline{\alpha}_{MP}}+\lambda\tilde{\mathbf{A}}_{\mathbf{W}}) = \frac{r}{2}\log\det(\gamma\mathbf{A}_{\mathbf{X}}|_{\underline{\alpha}_{MP}}+\lambda\mathbf{A}_{\mathbf{W}})+ \\ +\frac{N_C}{2}\sum_{i=1}^{r}\log\mu_i^{\mathbf{K}}\end{aligned} \tag{6.36}$$

$$\begin{aligned}\frac{1}{2}\log\text{pdet}\,\gamma\tilde{\mathbf{A}}_{\mathbf{X}}|_{\underline{\alpha}_{KMP}} = \frac{rN_C}{2}\log(\gamma)+\frac{r}{2}\log\det\mathbf{A}_{\mathbf{D}}|_{\alpha_{KMP}}+ \\ +\frac{N_C}{2}\sum_{i=1}^{r}\log\mu_i^{\mathbf{K}}\end{aligned} \tag{6.37}$$

$$\frac{1}{2}\log\text{pdet}\,\lambda\tilde{\mathbf{A}}_{\mathbf{W}} = \frac{rN_C}{2}\log(\lambda)+\frac{r}{2}\log\det\mathbf{A}_{\mathbf{W}}+\frac{N_C}{2}\sum_{i=1}^{r}\log\mu_i^{\mathbf{K}} \tag{6.38}$$

Substituting Eq.6.36-6.38 into Eq.6.35, it reduces to the equation for the approximate evidence in the primal space (i.e. Eq.6.34) where the dimensionality of the transformed space d' is replaced by the rank r of $\mathbf{K}$.

It is worth noting that r is the only spectral property of the kernel matrix that actually enters in the evaluation of the evidence. Indeed the sum of the log-eigenvalues of $\mathbf{K}$ represents a constant value with respect to γ and λ and can be safely ignored in the model selection phase. Interestingly, r can be interpreted as an estimate for the dimensionality of the underlying vector space. The numerical evaluation of such quantity however is not trivial due to the finite precision of float representations. With this respect in the experimental section we will show how the numerical evaluation of such parameter can be carried out setting the threshold for the non-zero eigenvalue to be: $\mu_x^K : \frac{\sum_{i=1}^{x} \mu_i^K}{\sum_{i=1}^{N} \mu_i^K} = 0.99$.

6.7 Discussion

In chapter 3 we introduced the concept of Principal Path in data space starting from the one of MFEP in statistical mechanics. Such link was reinforced and more formally formulated in this chapter by the newly introduced cost function Eq.6.3. This should be clear for example taking a look to the Plain Elastic Band method for finding Minimum Energy Path (MEP) introduced in chapter 3. Indeed there a chain of R simulation replicas were evolved according to the total potential energy:

$$\hat{U}(\underline{x}_1, \dots, \underline{x}_R) = \sum_{i=0}^{R} U(\underline{x}_i) + \frac{k}{2} \sum_{i=0}^{R-1} (\underline{x}_{i+1} - \underline{x}_i)^2$$

Comparing this equation to Eq.6.3 clearly highlights the connection with statistical mechanics. The proposed cost function can be interpreted as a Plain Elastic Band where the potential energy of the molecular system $\hat{U}(\underline{x}_1, \dots, \underline{x}_R)$ is replaced by the kernel k-means cost function $\Omega_X(\mathbf{W}, \mathbf{X})$ on the underlying data space X. A major difference here regards the nature of the two functions, indeed PEB deals with a potential energy function. Whereas the nature of the proposed cost, being defined over sampled data, is closer to the one of free energy.

We would like also to underline here a further connection to the field of statistical mechanics and more specifically to MD. Indeed one can easily verify that the maximization step of the approximate EM procedure up to $O(s^2)$ i.e. Eq.6.13 is equivalent to the string evolution equation for the String Method [49] discussed in chapter 3. Recalling the connection just made among Ω_X and the potential energy of a molecular system one can further speculate that the E-step of the proposed method is equivalent to the mean force estimate step of the SM as depicted in Fig.6.7. Interestingly in the work by Maragliano et al. the algorithm is only described in terms of Gradient Descent (GD) and a variational formulation is missing. A major difference with String Methods however is that, there the string evolution is performed online i.e. while the MD simulation is running here instead we deal a posteriori with the sampled data.

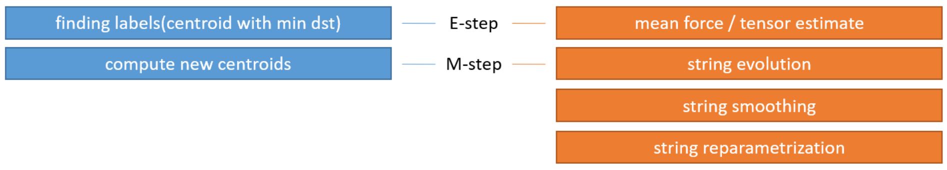

Figure 6.7: Schematics description of the connection between the standard EM procedure derived for our algorithm (in blue) and the string method for finding MFEP along MD simulations (in orange).

In the context of 1D manifold learning the proposed algorithm is closely related to the Elastic Map [56] framework discussed in chapter 3, indeed also there one defines a regularized functional form of the standard k-means cost. Our algorithm however presents some major differences such as: the presence of just one regularization parameter with no angular terms, the generalization to kernel space, the development of a Maximal Evidence principle for blindfolded in-sample model selection and the addition of fixed boundary conditions $\underline{w}_0$ and $\underline{w}_{N_C+1}$.

We would like to stress here this last aspect recalling that is particularly relevant especially in the model selection phase. Indeed adding fixed boundaries significantly reduces the space of available curves that can be produced by the algorithm. Moreover such boundary conditions allowed us to successfully apply the maximal evidence framework where a Bayesian interpretation of the parameters is straightforward. As a matter of fact we have shown that the proposed minimization primal problem can be rewritten as a maximum posterior problem where the prior is centered around a linear segment connecting the two endpoints. As it will be clear in the experimental section indeed a straight path connecting $\underline{w}_0$ to $\underline{w}_{N_C+1}$ is what is obtained if the regularization constant is pushed to infinity. In elastic maps, conversely, if the corresponding regularization constant is pushed to infinity the result is that all the clusters collapse in the same average point. Our approach is consistent with classical supervised learning methods where linear entities (linear separators, linear regressors) are the typical priors obtained in the limit of infinite regularization (for instance induced by the infinite variance of the gaussian basis functions). All these features allowed us to completely characterize the method in terms of hyperparameters.

Another less closely related method in the context of 1D manifold learning is the Self Organizing Map (SOM) [59] also discussed briefly in chapter

3. One indeed can use such method to obtain paths as well, however such family of algorithms deals with global manifold learning and more importantly does not have a functional formulation relying just on an optmization one.

Part III

Experiments

Chapter 7

Experiments on Clustering with DKK

The proposed Distributed Kernel K-means (DKK) method was completely implemented using C++, the distribution strategy was achieved via MPI (i.e. Message Passing Interface) and the offload acceleration was implemented with a combination of CUDA and OpenMP (to manage multiple threads on the host processor). In the following chapter we present a set of experiments where the algorithm was tested on datasets that are standard in the Machine Learning (ML) field as well as on 2D toy datasets in order to better understand its behavior. Lastly we challenge it against a real application scenario coming from the Molecular Dynamics (MD) domain.

We already pointed out while deriving the method in chapter 4 how the proposed algorithm is controlled via two fundamental parameters: B and s. We will investigate in this section how these parameters affect the approximation degree of the obtained clustering results. The other free parameters intrinsic to kernel k-means are the number of clusters N_C and the kernel parameter/s. In this regard we decided to run the algorithm with a Gaussian kernel of a given variance σ. For the considered datasets, where N_C is unknown a priori, we propose the use of an elbow criterion on the cost function (i.e. so as that an increase in N_C corresponds to a decrease of the cost function lesser than 1%). Regarding σ, we decided to work most of the time in a *linear regime* by setting it to 4 to 6 times the maximum pairwise distance in the dataset. In this way, we are alleviating the model selection phase and we are retrieving results which can be safely compared with the ones coming from linear k-means (see how k-means can be obtained as a limiting case of kernel k-means in chapter 2).

7.1 The datasets

For the sake of clarity we list hereafter all the datasets used in this experimental section together with their specifications.

2D toy Synthetic dataset containing 4 clusters of 10000 elements in a 2D feature space. Each cluster is generated by sampling a Gaussian distribution with center and width carefully selected in order to facilitate its visualization i.e. ($\sigma = [0.1, 0, 1], \mu = [0.5, 0.2]$), ($\sigma = [0.1, 0, 1], \mu = [0.8, 0.8]$) and ($\sigma = [0.1, 0, 1], \mu = [0.2, 0.7]$).

MNIST Standard dataset of handwritten digits used as benchmark for classification algorihtms [72]. It is composed by a training set of 60000 samples and a test set of 30000 samples. Each sample represents an image of 28 x 28 pixels in gray scale with 8-bit color depth. This accounts for a 784 dimensional feature space with integer features in the range $[0, 255]$. The only preprocessing we propose here is a normalization on the features obtained dividing them by 255.

Noisy MNIST Synthetic dataset in-house generated by starting from MNIST and adding uniform noise on 20% of the features randomly selected with uniform distribution . Each sample in the training set is perturbed 20 times in order to obtain a final dataset of 1200000 samples in a 784 dimensional normalized feature space.

RCV1 Reuters Corpus Volume I is a collection of manually labeled documents used as standard benchmark for classification in the domain of multilingual text categorization [73]. It is composed of 23149 training samples and 781265 test samples. Among the various formats available we used here its expression as normalized log TF-IDF (i.e. logarithmic term frequency-inverse document frequency) vectors in a sparse 47236 dimensional feature space. As already proposed in [74] we preprocessed the dataset removing samples with multiple labels and categories with less then 500 samples. After doing this we obtained a dataset of 193844 samples all coming from the test samples which we divided in 188000 training samples and 5844 test samples to maintain the original ratio. Moreover, to deal with the sparsity of the feature space we performed a dimensionality reduction step via random projection on a dense 256 dimensional space.

20 Newsgroup Another standard dataset in the field of text classification [75] is composed of 18828 documents divided into 11314 training samples and 7514 test samples evenly divided among 20 newsgroups. We expressed the dataset as normalized TF-IDF vectors in a sparse 101631 dimensional feature space. As already explained for the RCV1 dataset we deal with the sparsity of the feature space by means of a random projection on a dense 256 dimensional space. This is the only preprocessing needed for our purposes.

MD data As anticipated, we used MD as an appealing clustering scenario in which to leverage the features of the proposed algorithm. Microsecond-long trajectories of the binding mechanism of a drug, specifically a transition state analogue named DADMe-immucillin-H, to the Purine Nucleoside Phosphorylase (PNP) enzyme were employed [76].

7.2 The quality measures

When possible, we compared the clustering labels coming from the proposed procedure with the classification training labels. We will consider mainly the following two quality measures:

Clustering accuracy Let $\underline{u}$ be the set of labels obtained as a clustering result and let $\underline{y}$ be the set of the actual classes given as training or test. The clustering accuracy is defined as

$$\mu(\underline{y}, \underline{u}) = \sum_{i=0}^{N-1} \frac{\delta(\psi(y_i), u_i)}{N} \tag{7.1}$$

Where $\psi(u_i)$ is a mapping function which maps each clustering label to an actual training or test class. We propose here the use of a simple majority voting scheme to obtain such a mapping.

Normalized Mutual Information Let now be $n_i = \sum_{j=0}^{N-1} \delta(u_i, j)$, $m_i = \sum_{j=0}^{N-1} \delta(y_i, j)$ and $o_{i,j} = \sum_{k=0}^{N-1} \delta(u_k, i)\delta(y_k, j)$ the normalized mutual information is a quality measure defined as:

$$\mathrm{NMI}(\underline{y}, \underline{u}) = \frac{\sum_{i,j} o_{i,j} \log(\frac{N o_{i,j}}{n_i m_j})}{(\sum_i n_i \log(\frac{n_i}{N}))(\sum_i m_i \log(\frac{m_i}{N}))} \tag{7.2}$$

7.3 The platforms

We tested our implementation on a variety of different platforms in order to better describe the versatility and the potential impact of the proposed algorithm, we list them here:

IBM-BG/Q - Cineca/FERMI Cluster of 10240 computing nodes equipped with two octacore IBM PowerA2, 1.6 GHz processors each, for a total of 163840 cores. The available memory amounts to 16 GB / core and the internal network features a 5D toroidal topology.

IBM NeXtScale - Cineca/GALILEO Cluster of 516 computing nodes equipped with two octacore Intel Haswell 2.40 GHz processors for a total of 8256 cores. The available memory amounts to 8 GB / core and the internal network features Infiniband with 4x QDR switches.

State of the art Workstation Modern heterogeneous desktop machine equipped with two Intel E-6500 esacore processors, 64 GB of memory and a GTX 980 nVIDIA Graphic Processing Unit (GPU).

7.4 Assessing the degree of approximation

As a first step to assess the proposed clustering algorithm we consider the 2D toy dataset. We aim at better illustrating and helping the visualization of the evolution of the cluster centers along with the iterations of the outer loop. Incidentally, we want to highlight the consequences of a poor sampling strategy (concept-drift) and to give a rationale for understanding its quality.

In figure 7.1 the evolution of the cluster centers is followed for two different sampling strategies i.e. stride sampling and block sampling. Even though the final result is the same for such simple dataset it should be clear that the stride sampling strategy is superior in representing the structure of the dataset within each mini-batch. The underlying question is how could one assess the quality of the sampling strategy in a real case scenario where direct visualization is not possible. In figure 7.2(b) we try to answer by looking at the behaviour of the cluster center displacement. We can comment that if such quantity is constantly small with respect to the average cluster size, the mini-batches can be regarded as good representative of the entire dataset structure. In contrast, high values or spikes in the same quantity may reflect a poor sampling strategy.

Observing figure 7.2(a) we rather note that the inner loop of the proposed algorithm (i.e. the minimization of the partial cost $\Omega(\mathbf{X}^i, \mathbf{W})$) does indeed help minimize the global objective function $\Omega(\mathbf{X}, \mathbf{W})$.

We consider now the MNIST dataset in order to assess the degree of approximation introduced by the mini-batch approach and by the a priori sparse representation of the cluster centers. We ran our algorithm on the 60000 training samples of MNIST with $B = [1, 2, 4, 8]$, $s \in [0.025, 1.0]$ and we monitored the resulting clustering centers against the 10000 test samples in order to compute the clustering accuracy μ. Results as well as execution times are presented in figure 7.3. We observe that the algorithm is generally

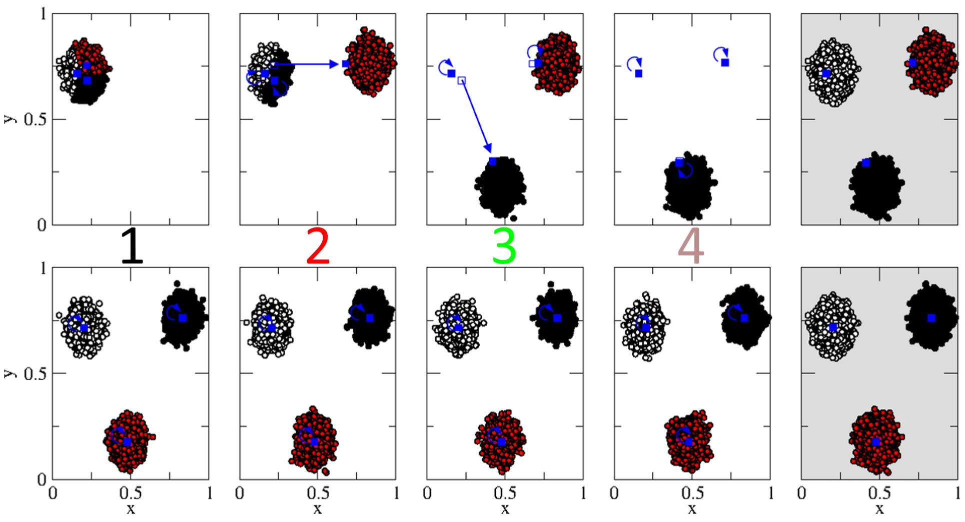

Figure 7.1: 2D Toy dataset. (top row) From left to right the evolution of the cluster centers across different iterations of the outer loop in the case of a poorly designed block sampling strategy. (bottom row) From left to right the evolution of the cluster centers across different iterations of the outer loop in the case of a proper stride sampling strategy where each mini-batch correctly captures the underlying structure of data.

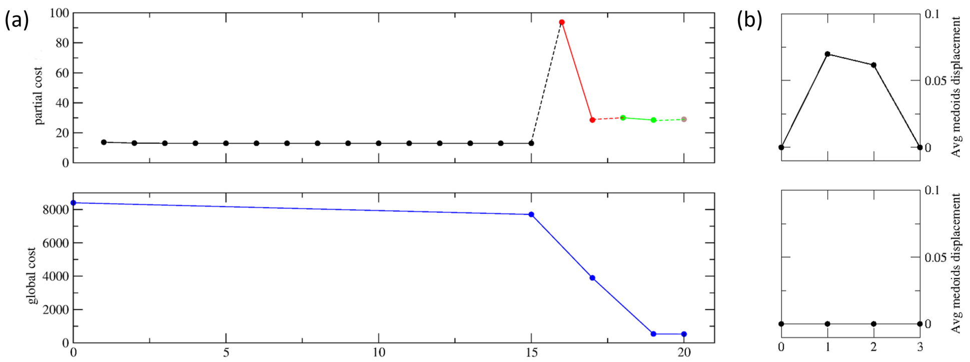

Figure 7.2: 2D Toy dataset. (a-top panel) Partial cost function $\Omega(\mathbf{X}^i, \mathbf{W}), \forall i \in [0, 3]$ vs number of iterations, different colors represent different mini-batches. (a-bottom panel) Global cost function Ω vs number of iterations. It is worth noting how the inner loop iterations within each mini-batch help to bring down the global cost function. (a) Average cluster centers displacement vs outer loop iterations for the two different sampling strategies illustrated in Fig.7.1, we propose this as a control observable to assess the quality of the sampling when direct visualization is not feasible.

robust across a wide range of the two parameters. The clustering accuracy slightly decreases when the number of mini-batches increase and once B is fixed it decreases almost monotonically with s dropping to low values when $s < 0.2$. As expected, this suggests us to position ourselves to the top-left part of the graph i.e. few mini-batches and $s \approx 1$.

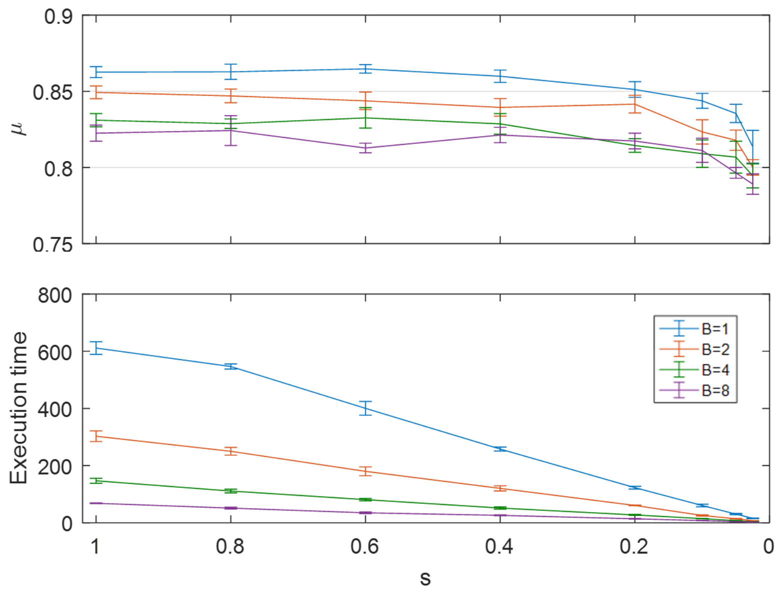

Figure 7.3: (top panel) Cluster Accuracy vs s. (bottom panel) Execution time vs s. Clustering performed on 60000 MNIST training samples evaluated against the 10000 provided test samples. Different colors represent different values of $B \in [1, 2, 4, 8]$. As described in the main text this graph can help understand how to perform model selection for the set of newly introduced parameters (B, s) picking a target execution time and looking at μ for the compatible sets of parameters.

Both B and s are trade-off parameters that have to be fixed. The strategy we suggest here is to fix a desired execution time on a given architecture. The available memory for the execution can lead to a first value for B using equation 4.16. As a starting point, the value of s can be fixed at its maximum. This set of parameters i.e. $(B_{\text{min}}, 1.0)$ should be optimal for the computational resources available i.e. minimum number of mini-batches without sparse representation of the cluster centroids. One can evaluate the expected execution time for the algorithm running it on a single mini-batch, if the expected execution time does not match the initial requirements then one can first slowly decrease s and, if this is not sufficient (i.e. expected execution time too high for $s < 0.2$), then increase the number of mini-batches. The approximation degree introduced can be self consistently checked using a single mini-batch and taking as a reference the results obtained for the optimal set of parameters $(B_{\min}, 1.0)$.

This rationale should guide the user to finely tune the trade-off parameters also on a large dataset.

7.5 Scaling behaviour

We aim here at assessing the original distribution strategy that we proposed in the previous section. In order to do so we tested our algorithm both on the IBM BG/Q and on the IBM NeXtScale machines above described, against the standard MNIST dataset.

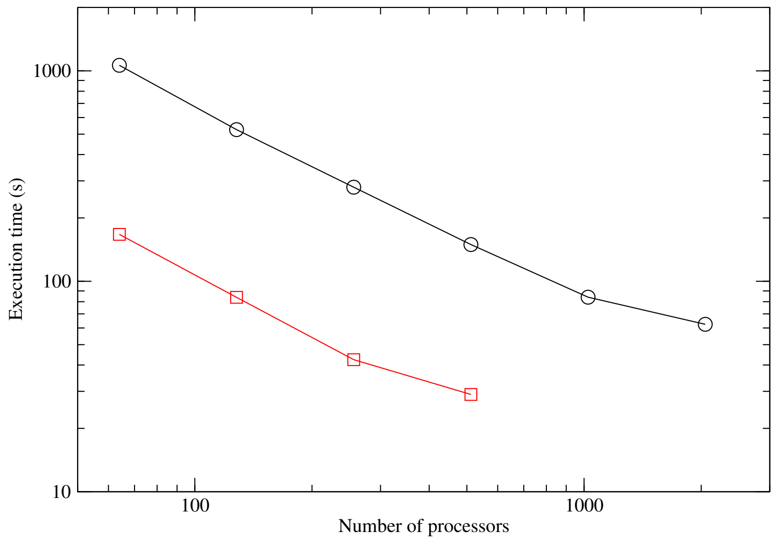

Figure 7.4: Execution time vs N_P for two different distributed architectures. IBM BG/Q in black/circles and IBM NeXtScale in red/squares.

We decided to set $B = 1$ in order to run the code in single batch mode since, as already explained, our distribution strategy does not involve the outer loop of the proposed method i.e. increasing the number of mini-batches would have only added a multiplicative constant to the execution time equal to B.

In Fig.7.4 the strong scaling plot for both machines is showed, the algorithm exhibits near to perfect scaling for a wide range of N_P i.e. $16 \rightarrow 1024$ on IBM BG/Q and $16 \rightarrow 256$ on IBM NeXtScale. The discrepancy from the ideal behaviour outside this range can be ascribed to the portion of code intrinsically serial (e.g. fetching and initialization phases) which becomes a prominent cost as described by Amdahl's law.

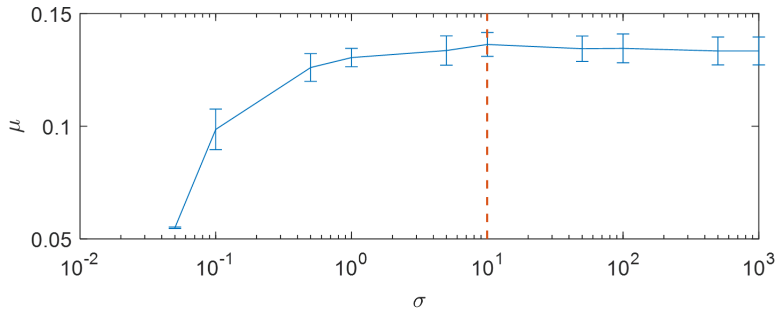

Figure 7.5: σ model selection for 20 newsgroup dataset. Maximum clustering accuracy obtained for $\sigma = 10$.

7.6 Standard datasets analysis

We present here the tests we performed on a state-of-the-art workstation over a standard dataset coming from the ML community. We show how even large datasets with up to 10^6 elements in 784 dimensions can be processed via a kernel approach on a desktop machine in a reasonable amount of time. The considered datasets are 20 Newsgroup (11314 samples in 256 dimensions), MNIST (60000 samples in 784 dimensions), noisy MNIST (1000000 samples in 784 dimensions) and RCV1 (188000 samples in 256 dimensions). The results are collected respectively in tables 7.1, 7.2, 7.3 and 7.4.

One can appreciate how for all the datasets the clustering accuracy decreases by less then 4% increasing the number of mini-batches from 1 to 4 while the overall execution time is reduced dramatically. As expected the clustering accuracy slowly decreases while further increasing the number of mini-batches B. However it should be noted how even for $B = 64$ one obtains qualitatively reasonable results i.e. with a reduction in the clustering accuracy of less than 10%.

In order to elucidate the model selection phase of standard kernel k-means parameters we show in Fig.7.5 and Fig.7.6 how to select σ on the 20 newsgroup dataset and how to select N_C on the original MNIST dataset.

As a baseline comparison for the clustering accuracy and the normalized mutual information we used a standard python implementation of k-means from the scikit-learn package [77]. Results coming from RCV1 are also compared with that appearing in the literature [74].

We close with a final note on the results obtained for the noisy MNIST dataset. Indeed we weren't able to run the standard implementation of k-means on the target workstation due to memory constraint therefore baseline comparison values for μ and NMI are not available. This however should also highlight the strength of the proposed algorithm in enabling kernel based clustering on large collection of data even on regular desktop machines.

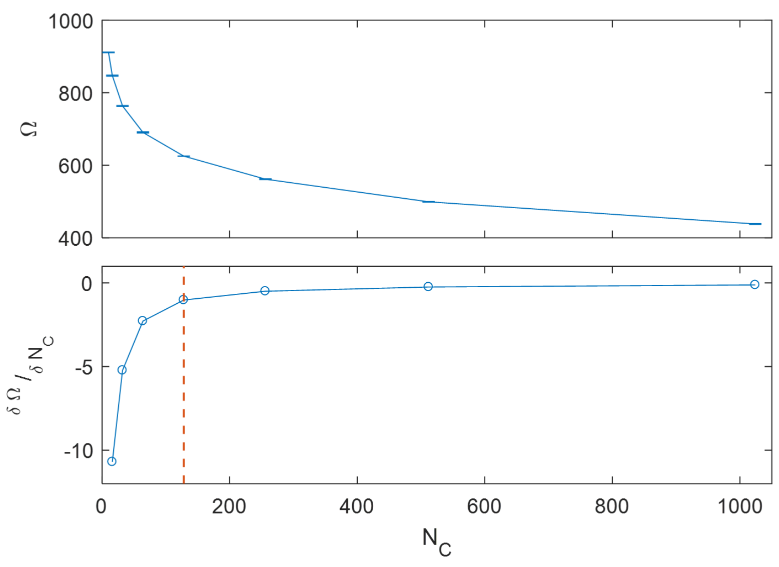

Figure 7.6: Cluster number model selection for MNIST original dataset. The elbow criterion indicated that $N_C = 128$ is a reasonable choice after which the numerical derivative $\frac{\delta\Omega}{\delta N_C}$ drops to zero.

$\tilde{N}$	μ	NMI	Execution time
Baseline	13.56 ± 0.66	0.075 ± 0.005	–
1	13.66 ± 0.39	0.068 ± 0.002	15.04 ± 0.88
4	12.72 ± 0.38	0.055 ± 0.004	6.31 ± 1.12
16	9.90 ± 0.56	0.051 ± 0.004	4.63 ± 0.74
64	8.92 ± 0.24	0.050 ± 0.002	4.15 ± 2.20

Table 7.1: 20 Newsgroup results and timings for different $\tilde{N}$

$\tilde{N}$	μ	NMI	Execution time
Baseline	84.5 ± 0.62	0.693 ± 0.012	–
1	86.47 ± 0.37	0.737 ± 0.006	655.23 ± 82.92
4	82.63 ± 0.91	0.680 ± 0.011	133.63 ± 4.40
16	81.45 ± 0.653	0.670 ± 0.010	32.17 ± 2.48
64	78.39 ± 0.95	0.626 ± 0.015	9.51 ± 0.58

Table 7.2: MNIST results and timings for different $\tilde{N}$

$\tilde{N}$	μ	NMI	Execution time
Baseline	–	–	–
32	64.19 ± 1.03	0.541 ± 0.005	2334.31 ± 25.63
64	60.97 ± 0.3	0.506 ± 0.001	1243.81 ± 23.43

Table 7.3: Noisy MNIST results and timings for different $\tilde{N}$

$\tilde{N}$	μ	NMI	Execution time
Literature	16.59 ± 0.62	0.2737 ± 0.0063	–
Baseline	15.16 ± 0.81	0.091 ± 0.0052	–
4	17.41 ± 0.83	0.147 ± 0.006	797.65 ± 53.48
16	16.52 ± 0.74	0.145 ± 0.001	170.96 ± 4.94
64	16.15 ± 0.60	0.132 ± 0.001	77.20 ± 3.96

Table 7.4: RCV1 results and timings for different $\tilde{N}$

7.7 Comparison with mini-batch SGD

As discussed in chapter 4 a closely related mini-batch technique already available in literature is the mini-batch Stochastic Gradient Descent (SGD) version of k-means proposed by Sculley [40]. A comparison about the clustering accuracy achieved by the two algorithms for the original MNIST dataset is shown in Fig.7.7. It is worth noting that the proposed DKK algorithm performs better as the number of minibatches B decreases whereas the performances of the SGD procedure proposed by Sculley are almost constant. Moreover, and as expected, our algorithm is less sensitive to noise, indeed the clustering accuracy variance is much lower in comparison to that of the SGC procedure.

In order to be as fair as possible about the comparison we used for both algorithms the same intial set of centroids obtained by a kernelized version of the k-means++ initialization scheme. Moreover since the output of our kernel based method is in the form of medoids $\mathbf{M}$ rather than actual cluster prototypes $\mathbf{W}$ we performed a set of experiments also applying a medoid approximation to the output of the SGD algorithm.

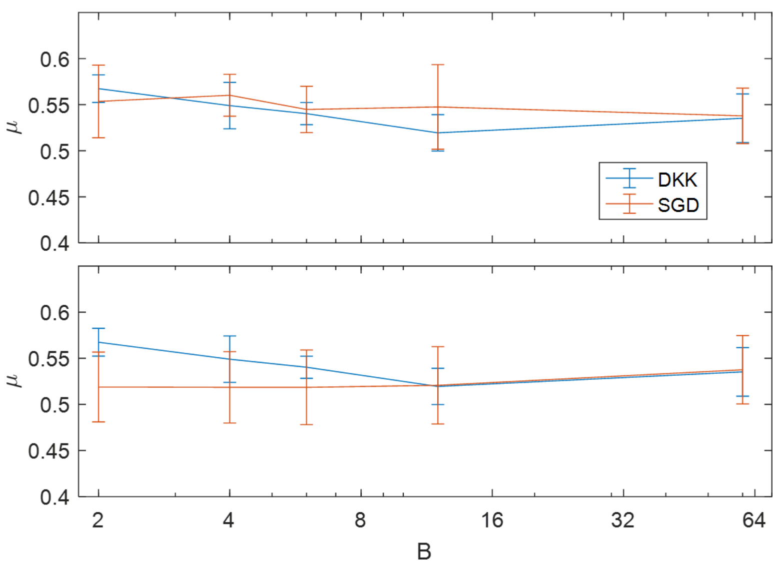

Figure 7.7: Clustering Accuracy μ vs number of mini-batches B for the proposed algorithm (blue line) and the SGD k-means procedure proposed by Sculley (red line). Comparison performed on the original MNIST dataset with $N_C = 10$, $\sigma = 4d_{\max}$ to mimic a linear behavior. Our algorithm was used to generate the initial cluster centroids that both algorithms used as starting points. Clustering accuracy evaluated on the original test set, the SGD performances were evaluated both without (top-panel) and with (bottom-panel) a medoid approximation on the output centroids.

7.8 MD application Scenario

As previously anticipated, we tested the proposed CUDA accelerated implementation of the algorithm on a real application scenario in the domain of MD. Microsecond-long trajectories of the binding mechanism of a drug, specifically a transition state analogue named DADMe-immucillin-H, to the PNP enzyme were employed. It is worth stressing the fact that such long trajectories well represent a good and relatively novel application domain for clustering and ML in general. The CUDA implementation enabled us to run the DKK on the entire dataset, consisting of 10^6 frames, in less than 20 seconds.

In this section we analyze the behavior of the clustering algorithm in terms of the quality of the obtained results in the MD domain. Basically, we compared the results obtained by the current implementation with respect to the results obtained in [1]. In that paper the binding process of a drug to its target was simulated and we used an in house clustering tool to get intermediate states of the protein/ligand complex formation along the binding routes. There, a k-medoids algorithm was used and the binding process was completely characterized.

Here we ran the same kind of analysis systematically verifying that the same, or very similar, intermediates could be obtained. For the analysis of the structures, we extracted the medoids from each cluster. The same atoms as per [1] were used for the clustering. To define the number of clusters we used the same elbow criterion as in [1] trying the clustering in the $(4, 40)$ range; in the end we obtained 20 clusters as an optimal value (see Fig.7.8).

For each run we initialized 5 times the algorithm with the kernel k-means++ method and kept the solution with minimum cost. To assess the accuracy of the approximated algorithm we split the dataset in 4 mini-batches each comprising about 250000 samples, thus drastically limiting the kernel matrix size with respect to a full run. We did not sparsify the clusters representation i.e. $s = 1$. We used the strided sampling because data was batch available and when possible this sampling should guarantee a propoer sampling of the underling data structure. As previously anticipated, we evaluated the quality of the results by the capability of the solution to capture the key events of the simulations. In Fig.7.9(a) we summarize the meaning of the medoids in structural terms using the same naming conventions appeared in [1] and associate them with the respective cluster id.

Overall those medoids well recapitulate the binding process giving the same synthetic description obtained in [74] despite the mini-batch approximation. In particular, we show Fig.7.9(b) the distance matrix computed across the medoids. We reordered the columns based on the manual classification induced by visual inspection. Results show clearly the three main macro-sections of the simulation namely the bound state, the entrance paths

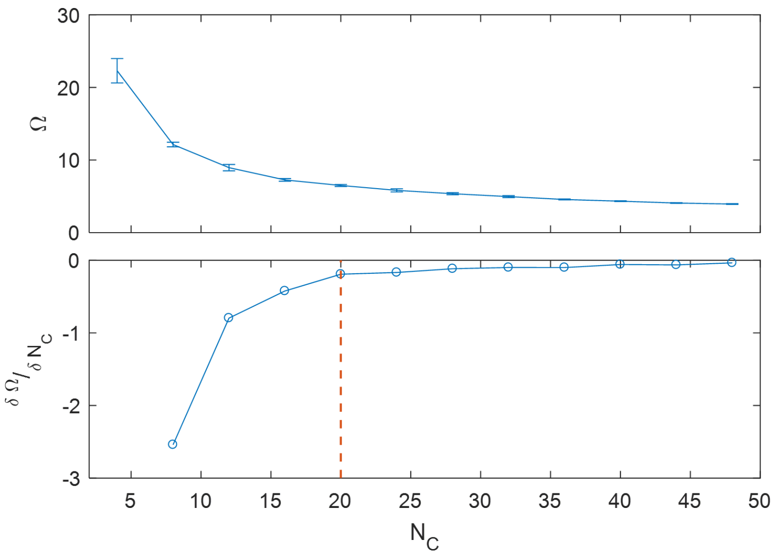

Figure 7.8: Clusters number model selection for molecular dynamics data. The elbow criterion indicated that 20 clusters were sufficient to correctly represent the data

(a)

Cluster Description	Cluster ids
Out	2,6,7,10,12,14,17,18
Gate	1,9,13
Upper	3,8
Frontal	5,19,16
Ensemble C	8,15,20
Ensemble B	4
Ensemble A	11

(b)

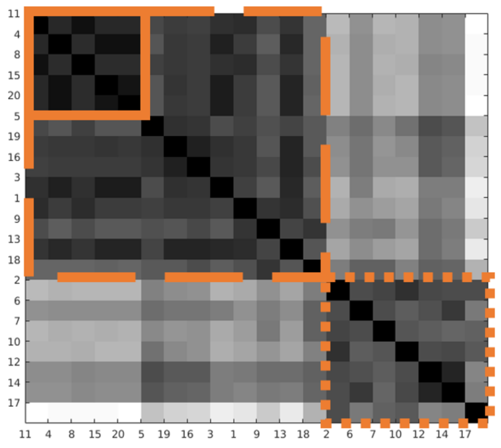

Figure 7.9: (a) Table summarizing medoids for MD data and their labeling (b) Medoids RMSD matrix. On the axis the medoid identifiers. On the upper left is well visible the macroarea of the bound states. Then, this area extends to the right including the entrance paths, and lastly, on the lower right corner, the unbound states.

Achieved FLOPs	2.6TFLOPS
Achieved occupancy	96%
Warp execution efficiency	99%
Shared memory efficiency	100%
Global memory load efficiency	100%
Global memory store efficiency	62%
Overall speed up (20 cores + GPU vs 20 cores)	$6.5\times$

Table 7.5: CUDA implementation performances - nVidia GTX980

and the out unbound states.

GPU performances

A detailed analysis of the algorithm in terms of major GPU performance metrics is presented in Table 7.5. Such performance analysis was done while the algorithm was clustering the MD dataset on a *state-of-the-arte* workstation (equipped with an nVidia GTX 980) using the visual profiler distributed by nVIDIA on an average of 60 CUDA kernel invocations. The efficacy of the proposed memory layout for the input trajectory should be clear given the high values of Shared memory efficiency and Global memory load efficency. The high value of achived occupancy also tells us that the latency due to memory load / store operations is successfully hided by the execution of concurrent warps. A comment has to be done on the relatively low value of the achieved Global memory store efficiency. Indeed one should consider to redesign the memory layout of the output kernel matrix K in order to maximize such value however the satisfactory results already obtained discouraged us to pursue such optimization.

Chapter 8

Experiments on the Principal Path Finding Algorithm

The proposed technique for finding Principal Paths in data space introduced in chapter 6 was implmented in MATLAB. In the following chapter a set of five experiments is presented in order to fully characterize such algorithm. Results will also prove how the concept of Principal Paths introduced in chapter 3 is robust and cognitively sound.

Synthetically generated bi-dimensional and tri-dimensional datasets are used in order to validate both the algorithm and the model selection procedure in a fully controlled environment. A comparison with Dijkstra's algorithm on the standard handwritten digits dataset MNIST follows, highlighting the energy relevance of the path found by the proposed method with respect to the one usually found by shortest path approaches. The noise robustness of the algorithm is then assessed against a noisy version of the Sheffield Face Dataset. As already anticipated in chapter 6 we then compare the proposed method with the closely related manifold-learning techniques of one-dimensional Self Organizing Map (SOM) and one-dimensional Elastic Map on dynamical systems exhibiting chaotic behaviour underlying the stability of the manifold captured by our algorithm. To conclude,a real application scenario coming from the Molecular Dynamics (MD) domain is presented.

The algorithm was run in its exact version i.e. solving the Toeplitz system of equations appearing in the M-step by direct inversion of the $\mathbf{A}$ matrix. We heuristically set the filter parameters to: $N_f = 200$, $k = 5$, $T = 0.1 \max_{i,j} d(\underline{m}_i, \underline{m}_j)$, a choice that proved to work on all the tested datasets. Moreover to avoid a bi-dimensional search for the global maximum of Bayesian evidence we also decided to heuristically set the value of $\frac{1}{\gamma}$ being driven by the simple Bayesian interpretation of the parameter as described in the appendices. Hereafter all the evidence curves are therefore plotted against the single regularization parameter $s = \frac{\lambda}{\gamma}$.

8.1 The datasets

For the sake of clarity we list hereafter all the datasets used in this experimental section together with their specifications.

Toys Several synthetic dataset will be used:

- A gaussian bi-dimensional model with 1500 samples drawn from 4 gaussians having the following parameters: $(\sigma = [0.2, 0, 2], \mu = [0, 0])$, $(\sigma = [0.1, 0, 1], \mu = [0, 0.3])$, $(\sigma = [0.05, 0, 05], \mu = [0.25, 0.75])$ and $(\sigma = [0.1, 0, 1], \mu = [0.3, 0.6])$.
- A noisy bi-dimensional circle dataset with 1500 samples generated adding a gaussian noise of variance $\sigma = 0.2$ on a circle of radius 1.
- A noisy bi-dimensional sinusoidal dataset with 1500 samples generaed adding a gaussian noise of variance $\sigma = 0.2$ on a sin function evaluated between $(0, 2\pi)$.
- A curl tri-dimensional dataset with 1500 samples generated by adding a gaussian noise of variance $\sigma = 0.2$ on a curve parametrized as $(cos(t), sin(t), t),\ t \in (0, 2\pi)$

MNIST Standard dataset of handwritten digits used as benchmark for classification algorihtms [72]. It is composed by a training set of 60000 samples and a test set of 30000 samples. Each sample represents an image of 28 x 28 pixels in gray scale with 8-bit color depth. This accounts for a 784 dimensional feature space with integer features in the range $[0, 255]$. The only preprocessing we propose here is a normalization on the features obtained dividing them by 255.

Sheffield Face Dataset Standard dataset fo face orientation detection [78]. Data consist of 564 images of 220 x 220 pixels with 256-bit grey-scale. Images of 20 different individuals, each in a range of poses from profile to frontal views.

Lorenz Noisy Trajectory A tri-dimensional synthetic dataset with 1500 samples generated by numerical integration of the well known Lorenz system of equation [79] :

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = x(\rho - z) - y \\ \dot{z} = xy - \beta z \end{cases}$$

where a gaussian noise of variance $\sigma = 0.1$ was added to the exact trajectory.

Dimensionless Chua Noisy Trajectory A tri-dimensional synthetic dataset with 1500 samples generated by numerical integration of the well known Chua dimensionless system of equations [80]:

$$\begin{cases} \dot{x} = \alpha(y - \theta(x)) \\ \dot{y} = x - y + z \\ \dot{z} = -\beta y \end{cases}$$

where a gaussian noise of variance $\sigma = 0.1$ was added to the exact trajectory.

8.2 A quality measure for Principal Paths

In order to define a proper quality measure for Principal Paths according to what we have discussed in chapter 3 we start from the notion of Principal Curve (PC) as originally introduced by Hastie and Stuetzle in [52]. We recall that the output model of the proposed algorithm is a path in the form of a polygonal chain of segments $p_i; i \in [1, NC + 1]$ having the following definition:

$$\underline{p}_i(t) = t\underline{w}_{i+1} + (t-1)\underline{w}_i; t \in [0,1] \tag{8.1}$$

Starting from such path, as shown in Fig.8.1, we divide the data into Voronoi partitions defined as:

$$\begin{aligned} V_i &= \{\underline{x} \in \mathbf{X} | i = \arg\min_j d(\underline{x}, \underline{p}_j)\} \\ d(\underline{x}, \underline{p}_i) &= \inf_{t \in [0,1]} ||\underline{p}_i(t) - \underline{x}|| \end{aligned} \tag{8.2}$$

Now, being inspired by the k-segments algorithm for PCs [53] we can introduce the following k-segments scoring function:

$$\Omega_{\text{ks}} = \sum_{i=0}^{N_C+1} \sum_{\underline{x} \in V_i} d(\underline{x}, \underline{p}_i) \tag{8.3}$$

which is an extension of the k-means cost function where the distance of each point to its closest prototype is replaced by the distance of each point to its closest segment. It should be understood here that such scoring function embedding the notion of orthogonal projection of the data on the curve $p(t)$, naturally represents a good proxy for assessing how much the path is close to a local portion of the PC.

In the following we will take advantage of such quality measure also to validate the maximal evidence framework introduced. Indeed, even though such Bayesian framework represents a powerful method to perform in-sample model selection more conventional out-of-sample approach to select the regularization parameter $s = \frac{\lambda}{\gamma}$ are possible.

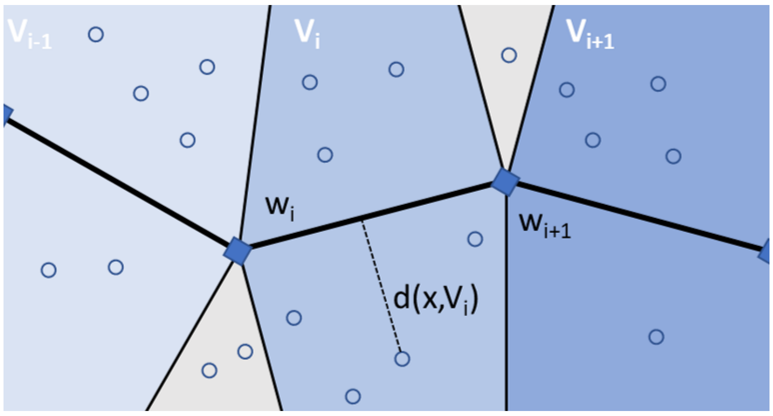

Figure 8.1: Visualization of the Voronoi regions implied by the polygonal chain path learned by the algorithm. The gray areas project onto a path vertex shared by two segments i.e. they can be assimilated to one of the two implied Voronoi regions without difference.

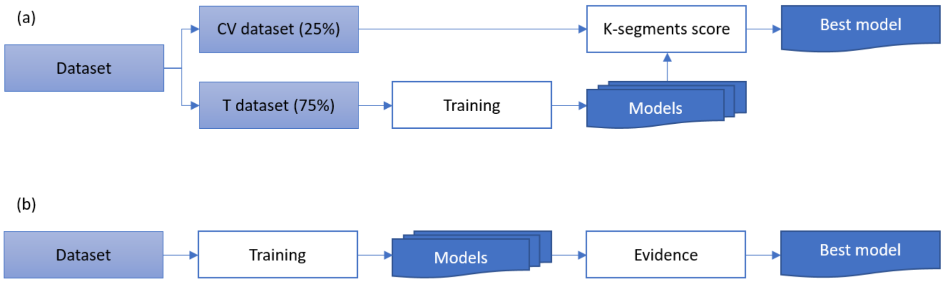

Figure 8.2: (a) Standard out-of-sample model selection scheme. (b) In-sample model selection scheme by means of Bayesian evidence maximization.

For example one could use standard Cross Validation to perform model selection as depicted in Fig.8.2(a). In order to do so, we can run the algorithm on a training set (i.e. dataset uniformly sub-sampled at 75%) obtaining several models for different values of s. The best model is then picked as the one that minimize the proposed k-segment scoring function on the cross validation set (i.e. remaining 25% of data).

It is worth noting that even though cross validation can always be used when the dataset is sufficiently large to be safely separated in a training set and a cross validation set, the in-sample ME approach should be preferred since it also offers a solid theoretical framework for model interpretation i.e. the Bayesian inference framework discussed before.

8.3 A note on the initialization of the algorithm

Being Ω a non-convex cost function to be optimized, the results of the algorithm heavily depend on the initialized paths. Hereafter in all the set of experiments we took advantage of a softening procedure in order to solve such issue. We train several models, starting with a high value of the regularization parameter s and slowly decreasing it. At each step of the procedure the algorithm is run up to convergence and the resulting path is used as initialization for the following iteration. The evidence of each model is evaluated using Eq.6.34 in order to obtain a model selection curve E vs s. The entire softening procedure together with an example of such evidence curve is detailed in Fig.8.3.

It is worth observing that such softening procedure is well motivated in the Bayesian framework previously introduced. Indeed setting s to a very high value and slowly decreasing it corresponds to start with a straight path connecting $\underline{w_0}$ to $\overline{\underline{w_{N_C+1}}}$ which slowly *feels* the data and relaxes towards them. i.e. We use as an initial guess for the path the center of the gaussian prior $P(\mathbf{W})$ as discussed in chapter 6.

8.4 Maximal Evidence Assessment

As a first assessment of the proposed maximal evidence framework we present the experiments performed in linear space on a bi-dimensional gaussian dataset keeping the boundary points fixed and increasing the number of clusters $N_C = \{10, 20, 40\}$. In-sample model selection via maximal evidence is compared with a standard out of sample cross validation minimizing the k-segment scoring function. Results are shown in Fig.8.4. Several comments can be done starting from such figure. First of all it should be noticed how the model selected by the two procedures are not only qualitatively similar but corresponds to similar values of s. So the Bayesian maximal evidence approach is able to select models which are close to those selected by a standard cross-validation technique without requiring held-out data. It is also possible to observe how the models selected by Bayesian inference are always smoother i.e. corresponds to slightly higher value of s. This should not be surprising since such Bayesian methods are usually known to exhibit an oversmoothing behavior.

Let us now comment on the stability of the results with respect to the usual k-means parameter N_C. Such parameter is of paramount importance while performing standard clustering and its selection phase is usually conducted via heuristic principle such as the elbow criteria used in the previous chapter. Here instead we show how the proposed regularization together with the maximal evidence model selection solve the problem in the case of Principal Path learning. Indeed as it is evident from the figure the paths

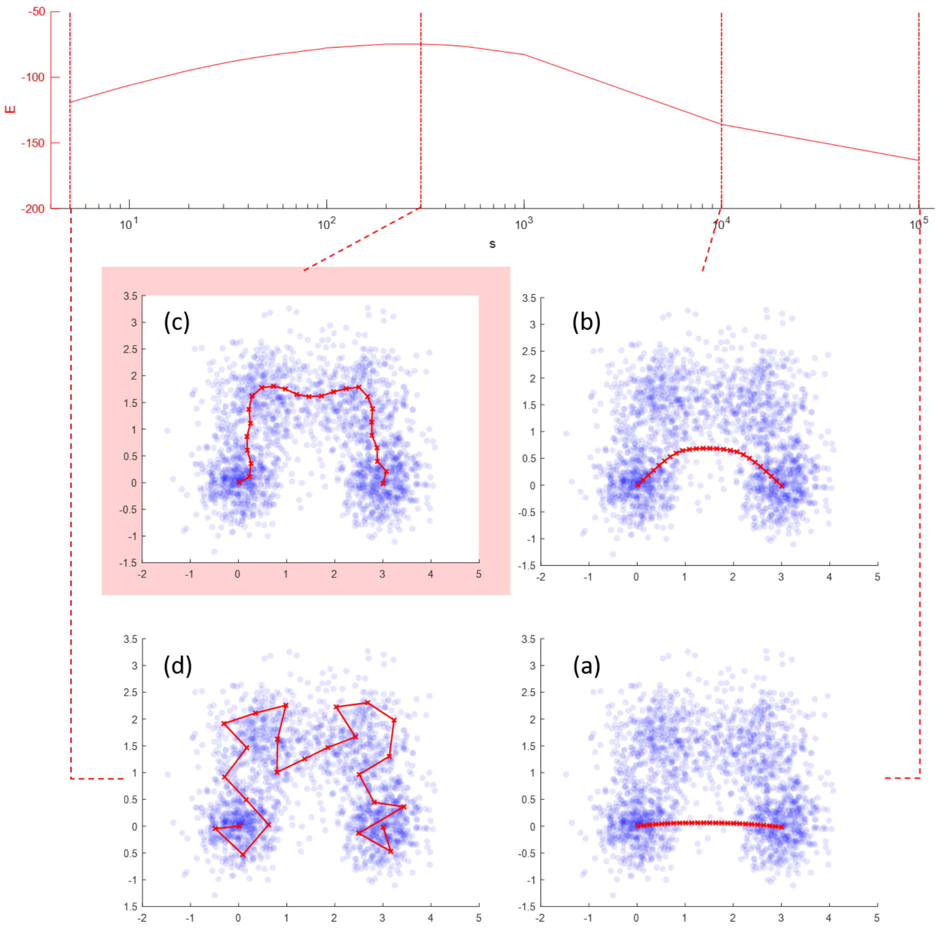

Figure 8.3: Schematic representation of the softening procedure used in the experiments. The procedure starts on the bottom-right panel (a) and proceed counterclockwise till (d). On the top panel an Evidence vs s curve is shown. One can appreciate how the max evidence solution (highlighted with a red background) is found before reaching too low values of s which would correspond to undesired rough paths.

found by the algorithm are stable across the whole range of N_C. It is interesting to notice how an increased number of clusters causes the peak of the evidence to shift right towards smoother solutions preventing the scattering of the cluster centers. The same analysis was conducted on several other bi-dimensional toy models always leading to good learned paths as depicted in Fig.8.5. The effectiveness of the proposed filter in selecting sub-manifolds of interest is also highlighted in Fig.8.5. For example the algorithm is able to focus on a semicircle in finding a path connecting two diametrically opposed points in the circular dataset.

We want now to discuss the validity of the kernel formulation of both the algorithm and the maximal evidence framework. For this purpose we run another set of experiments on a tri-dimensional curl dataset always keeping fixed the boundary conditions and increasing the number of clusters on a wider range $N_C = (10, 40, 80)$. The same considerations on the stability of the solution previously done holds true also in this case as should be clear by looking at Fig.8.7. As a second step we run the alorithm in kernel space with the same boundary conditions and for the same values of N_C. A gaussian kernel with variance $\sigma = 5d_{\max}$ was used to mimic a linear behavior (as already done in the previous experimental chapter on Distributed Kernel K-means (DKK)). The rank of the kernel matrix r was estimated forcing the percentage of variance on the non-zero eigenvalues of $\mathbf{K}$ to be 99% as depicted in Fig.8.8(a).

Comparing Fig.8.7 with Fig.8.8(b) one can qualitatively appreciate how the algorithm is able to retrive similar paths when running both in linear and kernel space. One should not be worried about the small fluctuations appearing in the kernel found paths. Indeed we stress the fact that in kernel space the output of the algorithm is given just in term of labels $\underline{u}$ and the path is visualized with a medoid approximation step. Moreover we stress the fact that the maximal evidence approach was able to retrive the exact same values of s_{ME} in both spaces thus validating the theoretical derivation detailed in chapter 6.

A more quantitative comparison is illustrated in Fig.8.6 where the evolution of the algorithm both in linear and kernel space is compared at each iteration. The average displacement between the centroids found in linear space and the medoids found in kernel space is rapidly going to almost 0 values showing how the algorithm is able indeed to retrieve the same model in both scenarios. We stress the fact that the large discrepancies appearing for high values of s should be ascribed to the representation problem in kernel space and the fluctuations on the labels to the finiteness of σ.

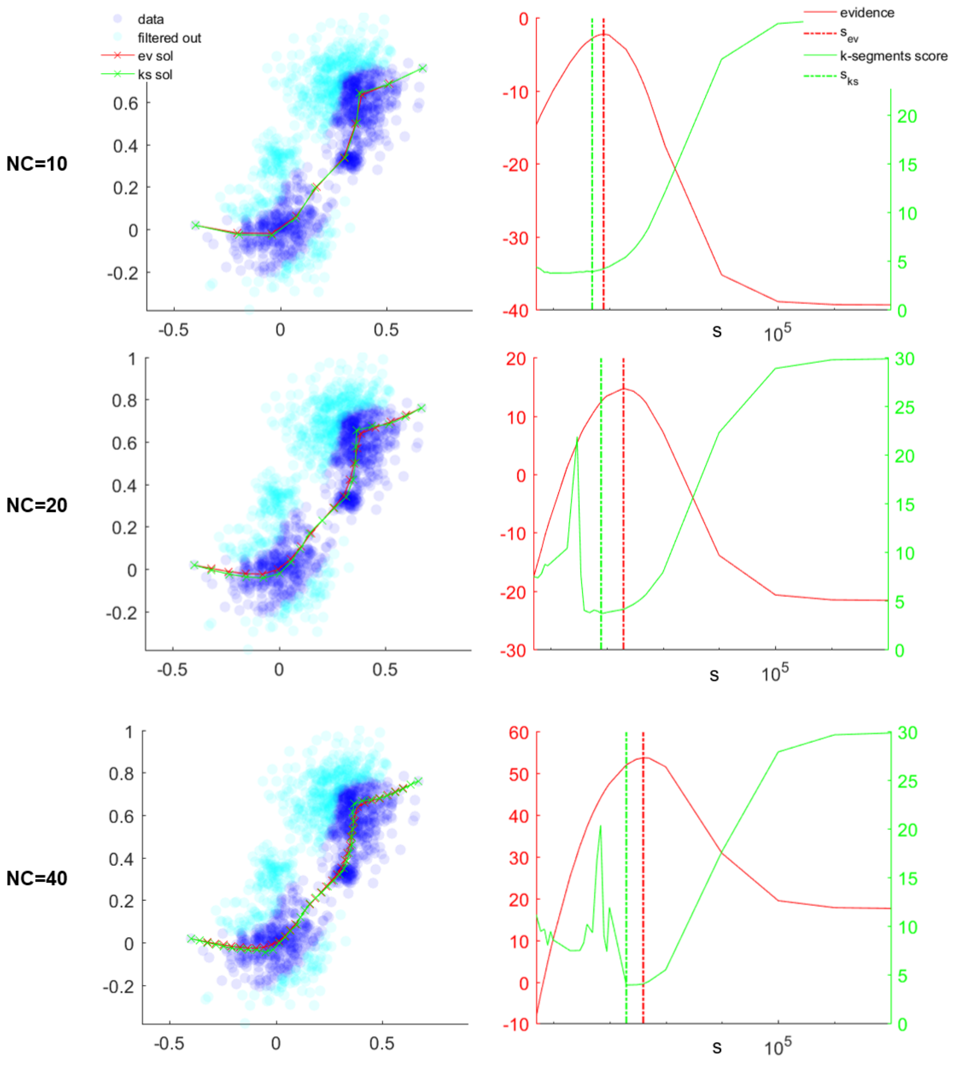

Figure 8.4: Results obtained for the same gaussian 2D toy dataset with increasing number of clusters N_C as specified. The stability of the algorithm and of the ME model selection phase should be appreciated observing how the obtained path does not change qualitatively with different values of N_C. It is also worth noting how Bayesian evidence has a smoother behaviour for large values of NC with respect to the proposed k-segment scoring function thus making it a better choice for blind-folded model selection.

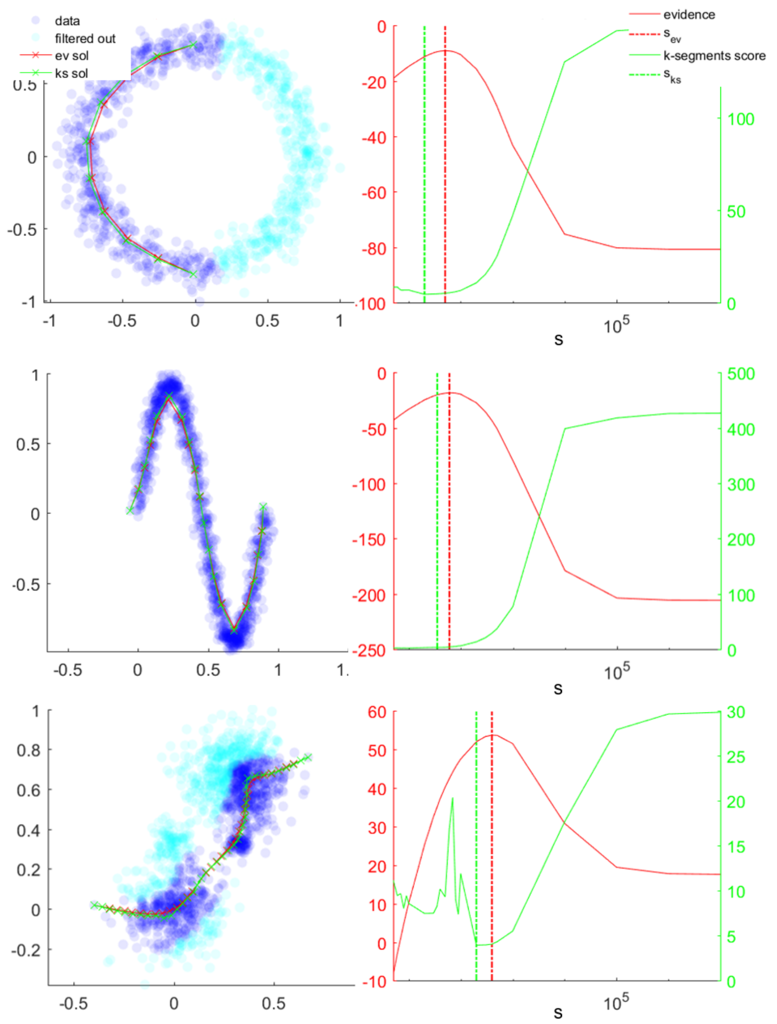

Figure 8.5: Results obtained for three different bi-dimensional toy datasets. On the right both filtered out data (light blue) and kept data (dark blue) are visualized together with the inferred models. On the left model selection curves for the proposed max-evidence approach (red) and a standard cross-validation technique with k-segment scoring function (green) are shown. The number of clusters N_C is fixed to 20.

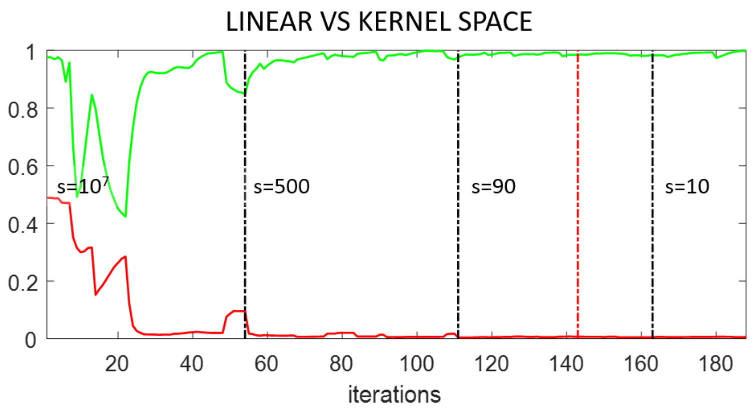

Figure 8.6: Comparison of the algorithm in linear and kernel space on the tri-dimensional curl dataset. Both fraction of equal labels (green) and the average centroids displacement (red) are plotted against the iterations of the algorithm. Vertical dotted lines highlights how the value of s is decreasing across iterations due to the softening procedure used. The red vertical dotted lines represents the iteration where the maximal evidence model was found.

8.5 A comparison with shortest path

Here we present the results obtained on the standard MNIST dataset of handwritten digits [81] aimed at elucidanting the difference between the usual shortest path and a principal path. The purpose of the experiments here is thus to asses the ability of the algorithm in finding energetically minimal transition paths. In doing so we aim at describing a morphing process leading, from a given starting sample representing a digit, to a final sample representing a different one. We run the algorithm in linear space (to facilitate visualization) with $N_C = 18$, using the softening procedure described before and selecting the best model via the maximal evidence approach. The value of γ was heuristically set so that $\frac{1}{\gamma} = \frac{\bar{d}}{N}$. In Fig.8.9 we show three of such models, for three different morphing processes i.e. $6 \to 3$, $7 \to 4$ and $6 \to 9$. It is interesting to notice how the inferred transition paths seems to embed the notion of morphing distance, minimizing the moves needed to convert the starting digit into the final one. For example in Fig.8.9(a) while converting a 6 into a 3, the algorithm suggests a transition path initially closing the upper-right part of the 6 to form an 8, later transforming it into a 3 by left-opening its double loop.

In order to better understand whether the inferred transition paths are energetically relevant we try to reconstruct the FES of the underlying morphing process as depicted in Fig.8.10. Given a model $\mathbf{W}$ we assume the data $\mathbf{X}$ to be generated by a gaussian process with probability $P(X|W)$ given by Eq.6.25. Let us use the following reaction coordinate to describe the samples

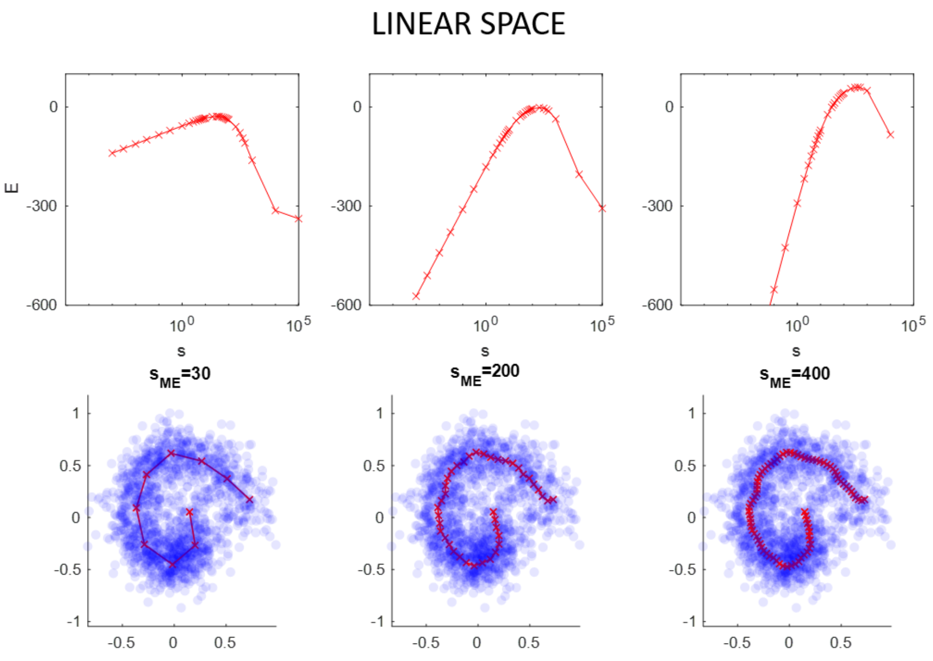

Figure 8.7: Principal paths found by the linear space algorithm on the tri-dimensional curl toy dataset together with the respective evidence vs s curves. The maximal evidence value s_{ME} is shown. From left to right the number of clusters increases in the range $N_C = (10, 40, 80)$. One can appreciate the stability of the algorithm and of the model selction phase against such parameter.

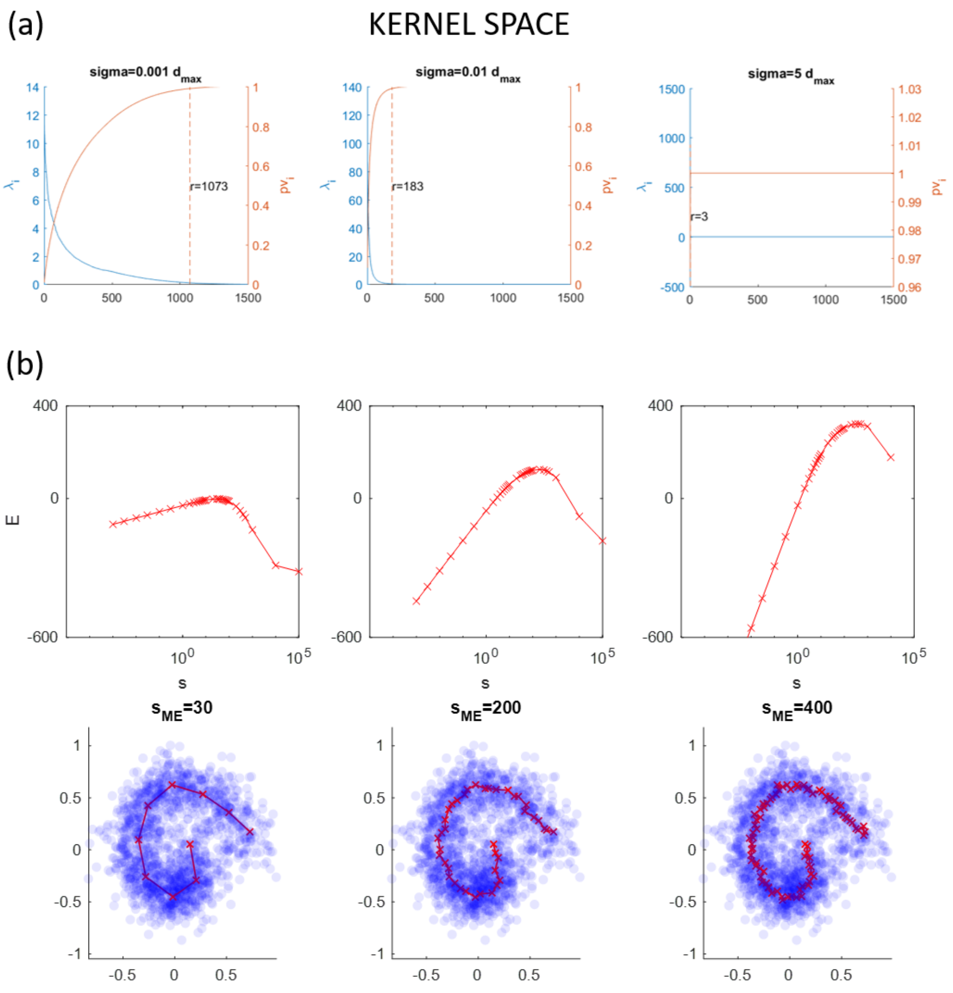

Figure 8.8: (a) Numerical estimation of r on a 3D dataset ($N = 1500$). Percentage of variance at 99% on the eigenvalues of $\mathbf{K}$ is used. Different graphs represents estimates for different value of the rbf kernel parameter σ. As expected the estimated dimensionality of the underlying input space decrease with the increase of σ from a value close to N to the actual value of the input when $\sigma = 5d_{\max}$. (b) Principal paths found by the kernel space algorithm on the tri-dimensional curl toy dataset together with the respective evidence vs s curves. The maximal evidence value s_{ME} is shown. From left to right the number of clusters increases in the range $N_C = (10, 40, 80)$.

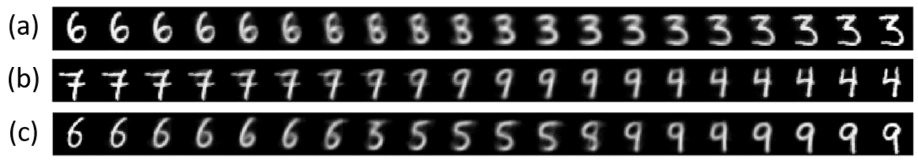

Figure 8.9: Three morphing paths inferred from MNIST handwritten digits for three different boundary conditions. Intermediate metastable states are visible in all three processes: (a)$6 \to 8 \to 3$ (b) $7 \to 9 \to 4$ (c) $6 \to 5 \to 9$.

generated by such process:

$$t(\underline{x}) = \frac{\sum_{j=0}^{i} \|\underline{w}_{j+1} - \underline{w}_j\| + \frac{\langle \underline{w}_i - \underline{x}, \underline{w}_{i+1} - \underline{w}_i \rangle}{\|\underline{w}_{i+1} - \underline{w}_i\|}}{\sum_j \|\underline{w}_{j+1} - \underline{w}_j\|}$$

. This reaction coordinate describes the evolution of the process along the manifold and represents a parameterization of the principal path starting from the source node and reaching the destination node. We can now estimate the states occupancy probability $p(t)$ as:

$$p(t^*) = \int d\underline{x} P(\underline{x}|\mathbf{W}) \delta(t(\underline{x}), t^*) \approx \frac{1}{N} \sum_{i=1}^{N} \delta(t(\underline{x}), t^*)$$

, the free energy $F(t)$ can be easily estimated as the negative logarithm of such occupancy probability.

We compare the implied FES for a path obtained by Dijkstra's shortest-path algorithm and a max-evidence path obtained by the proposed method for both a bi-dimensional toy Fig.8.11 and the MNIST dataset Fig.8.10. It should be clear how the proposed algorithm is superior in finding an energetically relevant description of the underlying process. Indeed in both examples the implied FES of the regularized path describe an energetically favorable process evolving through several meta-stable states separated by low energy barriers.

8.6 Assessing noise tolerance

In order to verify the robustness of the proposed algorithm against noise we conduct an experiment on the Sheffield Face Database. We artificially perturb the data introducing three levels of additive gaussian noise on 20, 40 and 80 percent of pixels. For each sample and for each noise level we generate 20 independent noisy samples. The algorithm was run in linear space and the value of γ was heuristically set so that $\frac{1}{\gamma} = \frac{\bar{d}}{N}$. As shown in Fig.8.12 the algorithm together with the max-evidence model selection

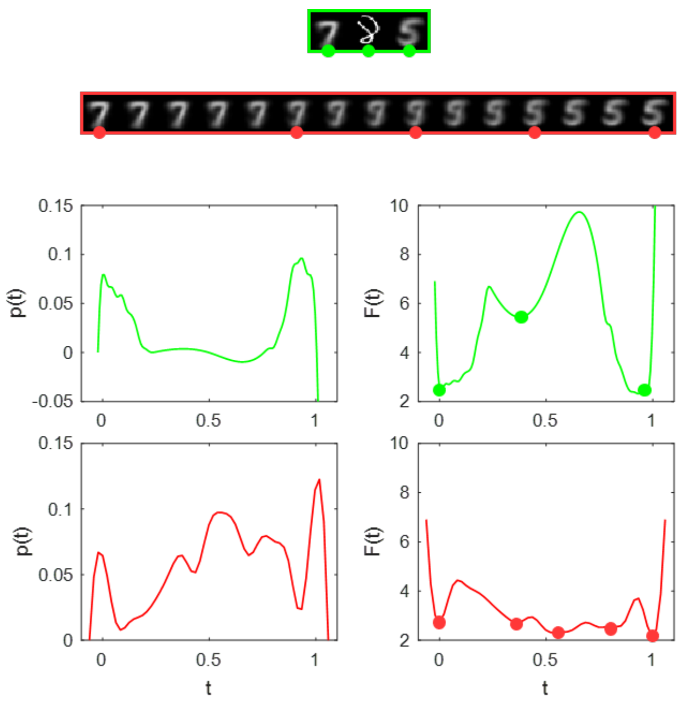

Figure 8.10: Reconstruction of the state occupancy probability $p(t)$ and of the FES $F(t)$ for a morphing path $7 \rightarrow 5$ inferred from the MNIST dataset of handwritten digits. Paths obtained by the proposed algorithm together with the max-evidence model selection approach (red) are compared with the shortest path algorithm connecting w_0 to w_{N_C+1} (green).

was able to reconstruct a meaningful transition path starting from a noisy profile-view sample to a noisy frontal-view sample of the same subject. We stress the fact that such result was obtained in the pixel space of the images without any pre-processing or featurization step.

Interestingly enough, the inferred transition path can be separated in three clear phases: a denoise of the profile view, the face rotation from profile to frontal view and a final noise injection to reach the end point of the path.

8.7 Manifold reconstruction

Lastly we validated the proposed technique against standard manifold learning algorithms in the description of strange attractors geometry.

We force the proposed algorithm to look for a closed path ($\underline{w}_0 = \underline{w}_{N_C+1}$) describing the entire dataset, that is without applying the filtering phase. The algorithm was run in linear space and the value of γ was heuristically set so that $\frac{1}{\gamma} = \bar{d}$.

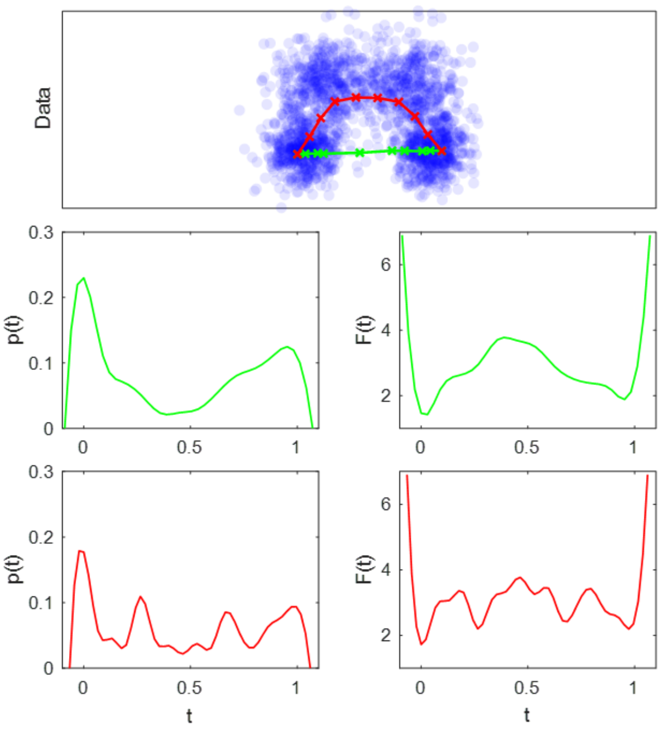

Figure 8.11: Reconstruction of the state occupancy probability $p(t)$ and of the FES $F(t)$ for a bi-dimensional toy model. Paths obtained by the proposed algorithm together with the max-evidence model selection approach (red) are compared with the shortest path algorithm connecting w_0 to w_{N_C+1} (green).

The inferred max-evidence path is compared with the one obtained by a one-dimensional toroidal SOM as implemented in [82] and the one obtained by a one-dimensional toroidal Elastic Map as implemented in [58]. Results for increasing values of N_C are shown in Fig.8.13. The three algorithms obtain relatively comparable results for low values of N_C but it should be clear how the proposed technique results more stable when the number of clusters increase. Indeed, thinking to the principal path as an invariant manifold, in Fig.8.13(c) one can appreciate how the length of the found paths by our algorithm remains constant for a wide range of N_C whereas this is not the case for the other methods.

8.8 MD application scenario

We close this section with an application scenario coming from the MD domain. The dataset is the same used in the previous chapter to test and

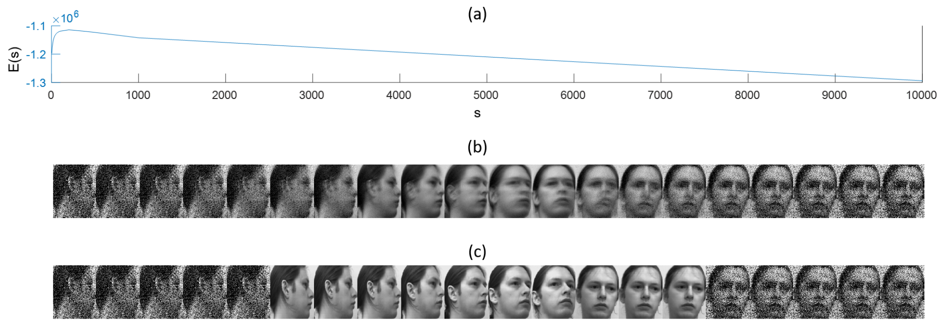

Figure 8.12: Model selection curve (a) together with the inferred transition path (b) and its medoids approximation (c) from a noisy profile-view image to a noisy frontal-view image in the Sheffield faces database.

validate the accelerated version of DKK. In this case we aim at describing the binding process of the drug namely DADMeimmucillin-H to its target, the Purine Nucleoside Phosphorylase (PNP) enzyme. The process was already studied in literature [1], thus allowing us to validate the obtained binding path. In Fig.8.14 three medoids of the inferred binding path are shown corresponding respectively to an unbound state, an intermediate entering state and the final bound state.

The found path is in good agreement with previous results, with the further significant advantage of having obtained this result in a completely automated way.

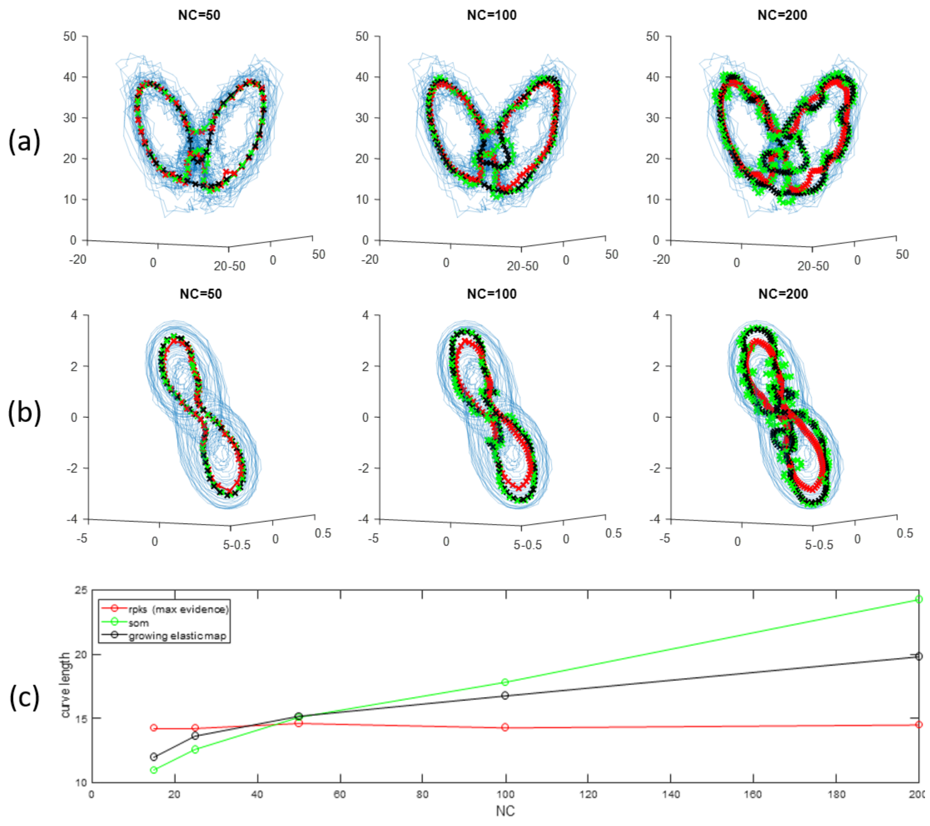

Figure 8.13: Inferred close paths describing the Lorenz attractor (a) and the dimensionless Chua attractor (b) for increasing values of N_C. (c) Simple generalization error measure i.e. curve length vs N_C. Results from the proposed algorithm (red), 1D SOM (green) and 1D Elastic Map (black) are shown.

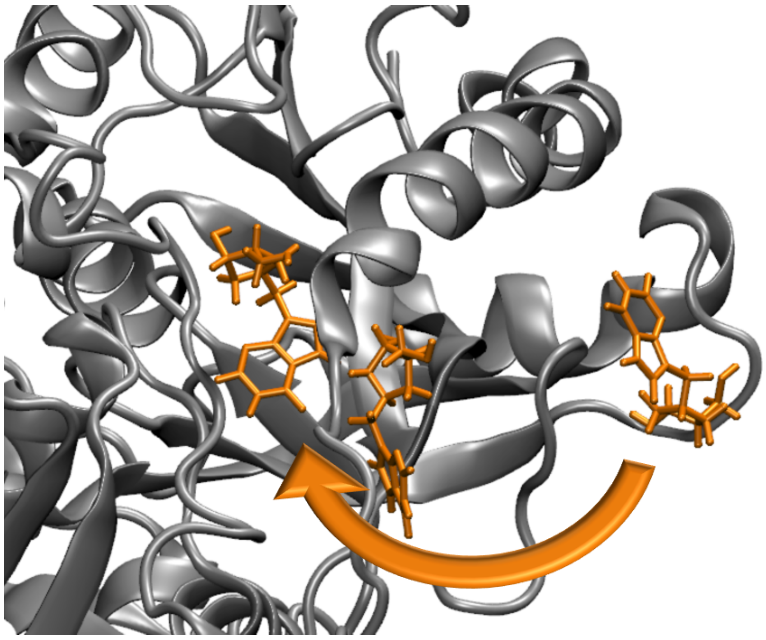

Figure 8.14: Pictorial description of the inferred binding path of the transition state analoge DADMeimmucillin-H to the PNP enzyme.

Part IV

Conclusions

Chapter 9

UL in the Context of MD: Findings and Perspectives

The thesis represented an extensive research in the multidisciplinary domain formed by the cross contamination of Unsupervised Learning (UL) and MD. We proved throughout the work how those two research fields are coming close creating a breeding ground for valuable new concepts and methods.

9.1 UL applied to MD: Distributed Kernel K-means

We discussed how Clustering can represent a valuable tool in the automatic analysis of long molecular trajectories in order to infer humanly interpretable models. With this respect we identified the kernelized version of k-means as a possible clustering algorithm meeting the special requirements of MD data. Starting from this consideration we therefore presented a novel engine to perform large scale kernel k-means clustering namely, DKK. We discussed how a two-fold approximation technique where the splitting of data into mini-batches is paired with a sparse representation of the cluster centroids can be used to effectively reduce the number of kernel matrix evaluations and how such approximation can be paired with an efficient strategy to scatter the computation across nodes in a distributed High Performance Computing (HPC) system. Remarkably, an adaptive strategy based on the available memory resources is proposed in order to naturally set the parameters controlling the two-fold approximation thus avoiding a burdening of the model selection phase. Moreover the nature of the proposed algorithm was exploited to accelerate the computation pairing the CPU with a general purpose GPUs (gpGPUs) and proposing an efficient CUDA implementation for the evaluation of Root Mean Square Deviation (RMSD) based kernel matrices. The resulting algorithm enabled us to run a set of experiments obtaining state of the art results not only in a real MD application scenario where μs long trajectories were successfully analyzed, but also on standard Machine Learning (ML) datasets where good clustering accuracy was achieved even in heavily approximated regimes. The set of experiments also highlighted a near to perfect strong scaling behavior of the algorithm on different HPC platforms, a particularly desirable feature in the big data era of ever growing datasets.

9.2 MD inspiring UL: Principal Paths

A second relevant portion of the thesis was devoted to the problem of finding Principal Paths in data space. The notion of Principal Path was at first introduced transposing the notion of Minimum Free Energy Path (MFEP) proper of statistical mechanics to the domain of Unsupervised data analysis. Discussing how the MFEP is approached within MD simulations we intuitively defined Principal Paths as smooth paths connecting a starting sample to an ending one, locally passing through the *middle* of the data. Such intuitive definition was then formalized introducing an actual functional to be minimized in the form of a regularized k-means cost. With this respect an EM-like optimization algorithm was derived both in linear and kernel space. Moreover the Bayesian framework of maximal evidence was successfully applied to the new learning problem obtaining operative equations in both spaces thus enabling in-sample model selection with the result of virtually having no free hyperparameter except the number of clusters (that represent a mere discretization step of the learnt curve).

We found, through several experiments, that the proposed functional coupled with such maximal evidence approach is able to systematically capture one dimensional manifolds that well fit the intuitive definition of Principal Paths. A particularly remarkable feature of the developed technique is its intrinsic robustness with respect to the number of clusters used to approximate the path, in this context we found the algorithm to be more stable with respect to other well known methods such the SOM or Elastic Maps. Interestingly enough we also found that the description implied by a Principal Path, gives insights into how to morph, at low energy cost, a starting data sample into an ending one thus validating the analogy with the MFEP.

9.3 Future perspectives

The first natural prosecution of the work presented here would be an extension of the DKK engine to find Principal Paths on large scale datasets. Indeed being the algorithm introduced for the principal path finding problem a regularized version of kernel k-means it could fit the distribution / acceleration scheme developed for DKK without requiring a significant computational overhead.

More significantly, being the proposed techniques kernel-based, several possibilities arise considering the variety of kernel functions and similarity measures that can be used. For example a particular appealing perspective, following the success of deep neural networks, would be exploring the possibility of deep kernels [83] in order to obtain a deep learning realization of the proposed algorithms.

9.4 A general remark about MD and UL

We close the thesis with a final, general remark about MD and UL. Let us observe how the central problem in MD can be identified with the one of FES reconstruction where one aims at studying a process of interest estimating the underlying FES. On the other hand the central problem of UL is the one of learning a *non trivial* representation of the data estimating the underlying manifold. It should therefore be clear how both the problems relates with the core problem of statistical inference i.e. estimating an unknown probability density (as a state occupancy probability in MD or as a Generative model in UL) starting from a set of observations.

We can conclude saying that the connection among MD and UL highlighted throughout the work is deep and has its foundation in the statistical nature at the heart of both fields.

Acronyms

CV Collective Variable.

DKK Distributed Kernel K-means.

EM Expectation Maximization.

FES Free Energy Surface.

GD Gradient Descent.

gpGPU general purpose GPU.

GPU Graphic Processing Unit.

HPC High Performance Computing.

MD Molecular Dynamics.

MEP Minimum Energy Path.

MFEP Minimum Free Energy Path.

ML Machine Learning.

MSM Markov State Model.

NG Neural Gas.

PC Principal Curve.

PNP Purine Nucleoside Phosphorylase.

RMSD Root Mean Square Deviation.

SGD Stochastic Gradient Descent.

SIMT Single Instruction Multiple Threads.

SOM Self Organizing Map.

UL Unsupervised Learning.

Glossary

A

accelerate sampling techniques techniques aimed at overcoming the problem of sampling rare-events (e.g. umbrella sampling, meta-dynamics, ...). An artificial bias potential is usually introduced to lower the energy barriers..

B

Boltzmann distribution Equilibrium probability distribution of the occupancy of micro-states for a closed system of fixed volume in thermal equilibrium with a heat bath..

C

canonical ensamble It represents the phase space of a system with fixed volume, in thermal equilibrium with a heat bath. A Molecular Dynamics simulation is assumed to sample the canonical ensamble when a proper thermostat is used..

centroid The centroid of a given set of samples is defined as that point in space obtained as the mathematical average of all those samples..

clustering Unsupervised Learning task of grouping unlabeled samples into groups according to a given similarity measure.

collective variable A collective variable is defined as a given function $\theta(\underline{x})$ of the system coordinates meant to be used as a reduced representation for the simulated process. It can be as simple as a dihedral angle, even though in real application scenarios more sophisticated and computationally expensive descriptors are often used..

conformational frame See frame (Molecular Dynamics).

D

dimensionality reduction Unsupervised Learning task of inferring a low dimensional embedding for the data samples.

E

empirical potential See force field.

ergodicity A process is said to be ergodic if the whole phase space is accessible and is explored during its time evolution so that thermodinamic average of quantities can be replaced with time average..

expectation maximization Iterative algorithm where a set of learnable parameters is optimized taking advantage of some hidden or latent variables. The following two steps are iterated up to convergence: an expectation step (E-step) where the latent variables are evaluated to compute the expected value of the objective function and a maximization step (M-step) where such expected value is maximized with respect to the learnable parameters..

F

force field (Molecular Dynamics) Functional form and set of parameters that allow the potential energy of a molecular system to be evaluated at each step of a Molecular Dynamics simulation. The set of parameters is usually empirical even though parameters from reliable quantum chemistry simulations can be exploited..

frame (Molecular Dynamics) Given a molecular system of N_a atoms, a conformational frame is the set of the $3N_a$ atom coordinates.

free energy (Helmholtz) Thermodinamical quantity mesuring the available work of a system at constant volume and in thermal equilibrium with a heat bath. Formally defined as $A = U - TS$ where U is the internal energy of the system, T the temperature and S the entropy. A spontaneous process at constant T is supposed to minimize A..

free energy surface Assuming a scalar collective variable z, the free energy surface of a system with respect to such collective variable is defined as $\mathcal{F}(z) = -k_B T \log P(z) + A$..

G

gradient descent Iterative optimization algorithm where an objective function is locally minimized with steps proportional to the negative gradient of the objective function at the current evaluation point..

H

hyper parameters Parameters of a learning machine that are not optimized during the training phase but are assumed to be known a priori. When the hyper parameters are not know an extra model selection phase is needed..

K

kernel methods (Machine Learning) Class of Machine Learning algorithms that do not require an explicit vector space for the data samples. A kernel matrix (i.e. similarity matrix) is instead sufficient to carry out the entire procedure.

kinetic model Model of a biochemical process (e.g. catalytic mechanism of an enzyme) as a series of macroscopic states and transition probabilities among them that relates to the actual reaction rates..

L

likelihood Probability of observing the data D given a set of parameters γ and a set of hyperparameters $\mathcal{H}$: $P(D|\gamma, \mathcal{H})$. It is usually seen as a function of the parameters to be maximized in order to obtain a useful learning paradigm..

M

manifold learning Unsupervised Learning task of inferring the low dimensional support of the probability distribution that generated the samples (assuming its existence).

medoid Closest data sample to a given centroid. Provided that the data are dense enough it can be used as approximation to the centroid in those cases where an explicit vector space is not available..

metastable state A metastable state corresponds to a local minimum of the free energy for a spontaneous process at constant volume and in thermal equilibrium with a heat bath..

model selection Selecting the best model with respect to the hyper parameters. Different models are usually trained for different values of the hyper parameters and the resulting models are scored by means of some rational..

O

overfitting Learning a model that matches too closely the available training data so that it fails to generalize and to make predictions on unseen data..

P

phase space The space of conformations available to the system (e.g. for a Molecular Dynamics simulation of a system with N_a atoms the phase space is represented by the $6D$ vector space of $\underline{x}_i, \underline{p}_i \forall i \in [1, N_a]$).

posterior Probability that a model parametrized by γ generated the data D provided the set of hyper parameters $\mathcal{H}$: $P(\gamma|D, \mathcal{H})$. Maximizing the posterior is a common learning procedure in the framework of Bayesian inference..

principal curve intuitively defined as smooth one-dimensional curve that pass through the *middle* of the data. It can be viewed as the result of a $1D$ manifold learning problem. .

prior Representing with γ the set of learnable parameters of a given model, the prior $P(\gamma)$ is the probability distribution expressing the available a priori knowledge on such set of parameters..

protein-ligand binding Process through which a small molecule (ligand) binds a portion of a target protein (binding site) by means of weak interactions. The binding event usually leads to a change of conformation for the target protein in order to serve a biological purpose (e.g. activation / inhibition of a biochemical reaction).

R

rare event (Molecular Dynamics) A transition between two metastable states separated by an energy barrier with $\Delta\mathcal{F} >> k_B T$. Indeed it is known from Arrhenius law that the probability of escaping a free energy minimum by thermal fluctuations is exponentially small with respect to the height of the barrier..

regularization Penalty on model's complexity usually introduced to prevent overfitting or smoother solutions..

S

stochastic gradient descent A gradient descent procedure where an empirical objective function obtained as a summation of single sample contributions is optimized one term at a time in stochastic order..

T

thermostat (Molecular Dynamics) Stochastic force added to the force field of a molecular system in order to simulate thermal fluctuations. It provides a way to thermalize the system thus allowing the simulation (under certain circumstances) to sample the correct equilibrium probability distribution of micro-states..

trajectory (Molecular Dynamics) Set of time-ordered conformational frames obtained via a Molecular Dynamics simulation..

List of Figures

Bibliography

[1] Sergio Decherchi, Anna Berteotti, Giovanni Bottegoni, Walter Rocchia, and Andrea Cavalli. The ligand binding mechanism to purine nucleoside phosphorylase elucidated via molecular dynamics and machine learning. *Nature communications*, 6, 2015.

[2] Ignasi Buch, Toni Giorgino, and Gianni De Fabritiis. Complete reconstruction of an enzyme-inhibitor binding process by molecular dynamics simulations. *Proceedings of the National Academy of Sciences*, 108(25), 2011.

[3] Ron O Dror, Albert C Pan, Daniel H Arlow, David W Borhani, Paul Maragakis, Yibing Shan, Huafeng Xu, and David E Shaw. Pathway and mechanism of drug binding to g-protein-coupled receptors. *Proceedings of the National Academy of Sciences*, 108(32), 2011.

[4] Vijay S Pande, Kyle Beauchamp, and Gregory R Bowman. Everything you wanted to know about markov state models but were afraid to ask. *Methods*, 52(1), 2010.

[5] Eric H Lee, Jen Hsin, Marcos Sotomayor, Gemma Comellas, and Klaus Schulten. Discovery through the computational microscope. *Structure*, 17(10), 2009.

[6] Hans C Andersen. Molecular dynamics simulations at constant pressure and/or temperature. *The Journal of chemical physics*, 72(4), 1980.

[7] Shuichi Nosé. A unified formulation of the constant temperature molecular dynamics methods. *The Journal of chemical physics*, 81(1), 1984.

[8] William G Hoover. Canonical dynamics: equilibrium phase-space distributions. *Physical review A*, 31(3), 1985.

[9] Daan Frenkel and Berend Smit. *Understanding molecular simulation: from algorithms to applications*, volume 1. Academic press, 2001.

[10] Lev Davidovich Landau and Evgenii M Lifshitz. *Statistical Physics: V. 5: Course of Theoretical Physics*. Pergamon press, 1969.

[11] Davide Branduardi, Francesco Luigi Gervasio, and Michele Parrinello. From a to b in free energy space. *The Journal of chemical physics*, 126(5), 2007.

[12] Svante Arrhenius. Über die reaktionsgeschwindigkeit bei der inversion von rohrzucker durch säuren. *Zeitschrift für physikalische Chemie*, 4(1), 1889.

[13] Alessandro Laio and Francesco L Gervasio. Metadynamics: a method to simulate rare events and reconstruct the free energy in biophysics, chemistry and material science. *Reports on Progress in Physics*, 71(12), 2008.

[14] Alessandro Barducci, Giovanni Bussi, and Michele Parrinello. Well-tempered metadynamics: a smoothly converging and tunable free-energy method. *Physical review letters*, 100(2), 2008.

[15] Glenn M Torrie and John P Valleau. Nonphysical sampling distributions in monte carlo free-energy estimation: Umbrella sampling. *Journal of Computational Physics*, 23(2), 1977.

[16] Yuqing Deng and Benoît Roux. Calculation of standard binding free energies: Aromatic molecules in the t4 lysozyme l99a mutant. *Journal of Chemical Theory and Computation*, 2(5), 2006.

[17] Jeff Wereszczynski and J Andrew McCammon. Statistical mechanics and molecular dynamics in evaluating thermodynamic properties of biomolecular recognition. *Quarterly reviews of biophysics*, 45(1), 2012.

[18] Vladimir Vapnik. *Statistical learning theory. 1998.* Wiley, New York, 1998.

[19] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7(Nov), 2006.

[20] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2), 2009.

[21] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1996.

[22] Theodoros Evgeniou, Massimiliano Pontil, and Tomaso Poggio. Regularization networks and support vector machines. *Advances in computational mathematics*, 13(1), 2000.

[23] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press, 2002.

[24] David JC MacKay. Bayesian interpolation. *Neural computation*, 4(3), 1992.

[25] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1. Oakland, CA, USA., 1967.

[26] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, volume 96, 1996.

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

[28] Jon M Kleinberg. An impossibility theorem for clustering. In *Advances in neural information processing systems*, 2003.

[29] Shai Ben-David and Margareta Ackerman. Measures of clustering quality: A working set of axioms for clustering. In *Advances in neural information processing systems*, 2009.

[30] Reza Bosagh Zadeh and Shai Ben-David. A uniqueness theorem for clustering. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence.* AUAI Press, 2009.

[31] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2), 1982.

[32] Leon Bottou, Yoshua Bengio, et al. Convergence properties of the k-means algorithms. *Advances in neural information processing systems*, 1995.

[33] Leon Bottou and Yoshua Bengio. Convergence properties of the k-means algorithms. *Advances in neural information processing systems*, 1995.

[34] Mark Girolami. Mercer kernel-based clustering in feature space. *IEEE Transactions on Neural Networks*, 13(3), 2002.

[35] Alex J Smola and Bernhard Schölkopf. *Learning with kernels.* GMD-Forschungszentrum Informationstechnik, 1998.

[36] Radha Chitta, Rong Jin, Timothy C Havens, and Anil K Jain. Approximate kernel k-means: Solution to large scale kernel clustering. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011.

[37] Leonard Kaufman and Peter Rousseeuw. *Clustering by means of medoids*. North-Holland, 1987.

[38] Hae-Sang Park and Chi-Hyuck Jun. A simple and fast algorithm for k-medoids clustering. *Expert systems with applications*, 36(2), 2009.

[39] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007.

[40] David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*. ACM, 2010.

[41] Nikolai Alex, Alexander Hasenfuss, and Barbara Hammer. Patch clustering for massive data sets. *Neurocomputing*, 72(7), 2009.

[42] Nikolai Alex and Barbara Hammer. Parallelizing single patch pass clustering. In *ESANN*, 2008.

[43] Stefan Faußer and Friedhelm Schwenker. Parallelized kernel patch clustering. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*. Springer, 2010.

[44] Gregory R Bowman, Vijay S Pande, and Frank Noé. *An introduction to Markov state models and their application to long timescale molecular simulation*, volume 797. Springer Science & Business Media, 2013.

[45] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004.

[46] R Elber and M Karplus. A method for determining reaction paths in large molecules: Application to myoglobin. *Chemical Physics Letters*, 139(5), 1987.

[47] Hannes Jónsson, Greg Mills, and Karsten W Jacobsen. Nudged elastic band method for finding minimum energy paths of transitions. In *Classical and quantum dynamics in condensed phase simulations*. World Scientific, 1998.

[48] Graeme Henkelman, Blas P Uberuaga, and Hannes Jónsson. A climbing image nudged elastic band method for finding saddle points and minimum energy paths. *The Journal of chemical physics*, 113(22), 2000.

[49] Luca Maragliano, Alexander Fischer, Eric Vanden-Eijnden, and Giovanni Ciccotti. String method in collective variables: Minimum free energy paths and isocommittor surfaces. *The Journal of chemical physics*, 125(2), 2006.

[50] Luca Maragliano and Eric Vanden-Eijnden. On-the-fly string method for minimum free energy paths calculation. *Chemical physics letters*, 446(1), 2007.

[51] Trevor Hastie. Principal curves and surfaces. Technical report, Stanford University of California, Laboratory for Computational Statistics, 1984.

[52] Trevor Hastie and Werner Stuetzle. Principal curves. *Journal of the American Statistical Association*, 84(406), 1989.

[53] Jakob J Verbeek, Nikos Vlassis, and B Kröse. A k-segments algorithm for finding principal curves. *Pattern Recognition Letters*, 23(8), 2002.

[54] Jakob J Verbeek, Nikos Vlassis, and Ben Kröse. A soft k-segments algorithm for principal curves. In *International Conference on Artificial Neural Networks*. Springer, 2001.

[55] Balázs Kégl, Adam Krzyzak, Tamás Linder, and Kenneth Zeger. A polygonal line algorithm for constructing principal curves. In *Advances in Neural Information Processing Systems*, 1999.

[56] Alexander N Gorban, Alexander A Pitenko, Andrei Y Zinovyev, and Donald C Wunsch. Visualization of any data with elastic map method. 2001.

[57] Alexander N Gorban and Andrei Y Zinovyev. Elastic maps and nets for approximating principal manifolds and their application to microarray data visualization. 2008.

[58] Alexander N Gorban, Balázs Kégl, Donald C Wunsch, Andrei Y Zinovyev, et al. *Principal manifolds for data visualization and dimension reduction*, volume 58. Springer, 2008.

[59] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1), 1982.

[60] Balázs Kégl, Adam Krzyzak, Tamás Linder, and Kenneth Zeger. Learning and design of principal curves. *IEEE transactions on pattern analysis and machine intelligence*, 22(3), 2000.

[61] Edgar Erwin, Klaus Obermayer, and Klaus Schulten. Self-organizing maps: ordering, convergence properties and energy functions. *Biological cybernetics*, 67(1), 1992.

[62] Rong Zhang and Alexander I Rudnicky. A large scale clustering scheme for kernel k-means. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 4. IEEE, 2002.

[63] Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming.* Addison-Wesley Professional, 2010.

[64] Douglas L Theobald. Rapid calculation of rmsds using a quaternion-based characteristic polynomial. *Acta Crystallographica Section A: Foundations of Crystallography*, 61(4), 2005.

[65] Berk Hess, Carsten Kutzner, David Van Der Spoel, and Erik Lindahl. Gromacs 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation. *Journal of chemical theory and computation*, 4(3), 2008.

[66] James C Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with namd. *Journal of computational chemistry*, 26(16), 2005.

[67] Víctor A Gil and Víctor Guallar. pyrmsd: a python package for efficient pairwise rmsd matrix calculation and handling. *Bioinformatics*, 2013.

[68] James Durbin. The fitting of time-series models. *Revue de l'Institut International de Statistique*, 1960.

[69] GY Hu and Robert F O'Connell. Analytical inversion of symmetric tridiagonal matrices. *Journal of Physics A: Mathematical and General*, 29(7), 1996.

[70] Sergio Decherchi, Mauro Parodi, and Sandro Ridella. Learning the mean: A neural network approach. *Neurocomputing*, 77(1), 2012.

[71] Alexander Graham. Kronecker products and matrix calculus: With applications (mathematics and its applications) pdf. 1981.

[72] Yann LeCun and Corinna Cortes. The mnist database of handwritten digits, 1998.

[73] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr), 2004.

[74] Wen-Yen Chen, Yangqiu Song, Hongjie Bai, Chih-Jen Lin, and Edward Y Chang. Parallel spectral clustering in distributed systems. *IEEE transactions on pattern analysis and machine intelligence*, 33(3), 2011.

[75] Thorsten Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. Technical report, Carnegie-mellon univ pittsburgh pa dept of computer science, 1996.

[76] Meng-Chiao Ho, Wuxian Shi, Agnes Rinaldo-Matthis, Peter C Tyler, Gary B Evans, Keith Clinch, Steven C Almo, and Vern L Schramm. Four generations of transition-state analogues for human purine nucleoside phosphorylase. *Proceedings of the National Academy of Sciences*, 107(11), 2010.

[77] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct), 2011.

[78] Daniel B Graham and Nigel M Allinson. Characterising virtual eigensignatures for general purpose face recognition. In *Face Recognition*. Springer, 1998.

[79] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of the atmospheric sciences*, 20(2), 1963.

[80] Leon O Chua, Chai Wah Wu, Anshan Huang, and Guo-Qun Zhong. A universal circuit for studying and generating chaos. i. routes to chaos. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 40(10), 1993.

[81] Yann LeCun, Corinna Cortes, and CJC Burges. The mnist dataset of handwritten digits. 1998.

[82] Juha Vesanto, Johan Himberg, Esa Alhoniemi, Juha Parhankangas, et al. Self-organizing map in matlab: the som toolbox. In *Proceedings of the Matlab DSP conference*, volume 99, 1999.

[83] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial Intelligence and Statistics*, 2016.