High Dimensional Data Clustering by means of Distributed Dirichlet Process Mixture Models

Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, Florent Masseglia

High Dimensional Data Clustering by means of Distributed Dirichlet Process Mixture Models

Khadidja Meguelati
Inria, LIRMM, Univ Montpellier, CNRS
Montpellier, France
khadidja.meguelati@inria.fr

Benedicte Fontez
MISTEA, Montpellier SupAgro, Univ Montpellier
Montpellier, France
benedicte.fontez@supagro.fr

Nadine Hilgert
MISTEA, INRA, Univ Montpellier
Montpellier, France
nadine.hilgert@inra.fr

Florent Masseglia
Inria, LIRMM, Univ Montpellier, CNRS
Montpellier, France
florent.masseglia@inria.fr

Abstract

Clustering is a data mining technique intensively used for data analytics, with applications to marketing, security, text/document analysis, or sciences like biology, astronomy, and many more. Dirichlet Process Mixture (DPM) is a model used for multivariate clustering with the advantage of discovering the number of clusters automatically and offering favorable characteristics. However, in the case of high dimensional data, it becomes an important challenge with numerical and theoretical pitfalls. The advantages of DPM come at the price of prohibitive running times, which impair its adoption and makes centralized DPM approaches inefficient, especially with high dimensional data. We propose HD4C (High Dimensional Data Distributed Dirichlet Clustering), a parallel clustering solution that addresses the curse of dimensionality by two means. First it gracefully scales to massive datasets by distributed computing, while remaining DPM-compliant. Second, it performs clustering of high dimensional data such as time series (as a function of time), hyperspectral data (as a function of wavelength) etc. Our experiments, on both synthetic and real world data, illustrate the high performance of our approach.

Index terms— Gaussian random process, Dirichlet Process Mixture Model, Clustering, Parallelism, Reproducing Kernel Hilbert Space

1 Introduction

Clustering is a data mining technique intensively used for data analytics, with applications in marketing [1], security [2], or sciences like astronomy [3] and many more. Clustering may be used for identification in the new challenge of digital agriculture, where large amounts of complex data are collected: for example in herd monitoring, animal activity is monitored using a collar-mounted accelerometer, as illustrated in figure 1. One of the main difficulties, for clustering, is the fact that we do not know, in advance, the number of clusters to be discovered. To help performing cluster analysis, despite the unknown tackled number of clusters, the field of statistics contains several suggestions:

Figure 1: An accelerometer mounted on a sheep's collar.

1. Setting a number of clustering runs, with varying value of cluster number, and selecting the one that minimizes a goodness of fit criteria. It may be a quadratic risk or the Residual Mean Squared Error of Prediction (RMSEP) [4]. This approach needs the implementation of a cross-validation algorithm [4]. The clustering approach may be a mixture model with an Expectation-Maximization (EM) algorithm [5], or K-means [4] for instance.
2. Performing a hierarchical clustering and then cutting off the tree at a given depth, usually decided by the end-user. Different approaches for pruning with advantages and drawbacks exist, see [4].
3. Using a Dirichlet Process Mixture (DPM) which automatically detects the number of clusters [6].

In this work, we focus on the DPM approach since it allows estimating the number of clusters and assigning observations to clusters, in the same process. Furthermore, its implementation is quite straightforward in a Bayesian framework. Such properties of DPM make it a very appealing solution for many use-cases.

Unfortunately, DPM relies on matrix computations and is highly time consuming, especially in the case of high dimensional data. Several attempts have been made to make it distributed, however these approaches are not adapted for high dimensional data (see the discussion in Section 3).

We propose HD4C (High Dimensional Data Distributed Dirichlet Clustering), a novel parallel clustering approach adapted for high dimensional data, based on a distributed algorithm for Dirichlet Process Mixture. HD4C takes advantage of the properties of Reproducible Kernel Hilbert Spaces to allow clustering on the whole data (the whole signal or curve or time series) [7]. Other approaches that use feature selection and/or dimensionality reduction (like PCA or SVM) are often inappropriate because clusters generally lie in different subspaces [8].

The paper is organized as follows. The problem is stated in Section 2 with the necessary background. In Section 3 the related work is discussed. Our distributed solution for high dimensional data clustering by means of Dirichlet Process Mixture is detailed in Section 4. The efficiency and effectiveness of our approach are illustrated in Section 5 through an experimental evaluation. Finally, the conclusion is in Section 6.

2 Problem Definition

The problem we address is as follows. Given a (potentially big) dataset of *records* find, by means of a parallel process, a partition of the dataset into disjoint subsets called clusters, such that:

- Similar records are assigned to the same cluster.
- Dissimilar records are assigned to different clusters.
- The union of the clusters is the original dataset.

3 Related Work

3.1 High Dimensional Data Clustering

We set our work in the context of high dimensional data clustering, which covers time series and functional data (signal) clustering.

For time series clustering, three categories are defined by [9]: "Whole time-series clustering", "subsequence clustering", and "time point clustering". Subsequence clustering is performed on a set of subsequences extracted via a sliding window from a single long time-series, Keogh and Lin [10] showed that this type of clustering is meaningless. Time-point clustering also is applied on a single time series, and it is similar to subsequences clustering. The focus of our work is "whole time-series clustering". The authors of [9] and [11] identified four different approaches to do this, respectively for time series and functional data:

1. Work directly with raw data,
2. Work indirectly with features extracted from the raw data. For example, in [12] a symbolic representation of time series called SAX is presented,
3. Use a specific distance or dissimilarity, like Dynamic Time Warping (DTW) for time series or RKHS properties for functional data (allows to define an inner product and therefore a distance in a specific space),
4. Build a model to estimate features from the data and to cluster simultaneously.

A K-means algorithm is often suggested with the first two identified approaches because of its fast convergence and its scalability (distributed versions of K-means for multivariate data are available). The third approach requires to adapt the K-means, which uses an Euclidean distance, to more complex cases. Time warp for temporal data does not define a true distance (no triangular inequality). [13] proposed a generalized K-means based clustering for temporal data under time warp, but no version for distributed data is available. In a broader context, a clustering method for misaligned curves was developed by [14] based on warping functions. But this approach assumes landmarks, or more generally known warping functions, to re-align the data and is not proposed for distributed data.

Finally, K-means algorithms for functional data were proposed by [11] but again the algorithms are not presented for distributed data.

Moreover, one drawback of the K-means [15] is that it requires the number of clusters k to be specified in advance. In comparison, the Dirichlet Process mixture (DPM) [6] approach automatically detects the number of clusters and distributed versions for multivariate data are now available [16–19]

3.2 Massive Datasets Clustering

There is a significant research on clustering of big data. Some efforts have focused on making the similarity measures faster, like, *e.g.*, Zhu et al. [20] who introduced a novel data-adaptive approximation to DTW which can be quickly computed. Other studies suggest to make the main clustering algorithms scalable by means of massive distribution.

In our work, we focused on algorithms inspired by the DPM which allows estimating the number of clusters and assigning observations to clusters, in the same process. Unfortunately, DPM is highly time consuming. Consequently, several attempts have been done to make it distributed. However, while being effectively distributed, these approaches usually suffer from convergence issues (imbalanced data distribution on computing nodes) [17,18], or do not fully benefit from DPM properties [19]. Furthermore, making DPM parallel is not straightforward since it must compare each record to the set of existing clusters, a highly repeated number of times. That impairs the global performance of the approach in parallel, since comparing all the records to all the clusters would call for a high number of communications and make the process impractical.

In [16] a distributed DPM algorithm called DC-DPM (Distributed Clustering by Dirichlet Process Mixture) was introduced. It allows each node to have a view on the local results of all the other nodes, while avoiding exhaustive data exchanges. The main novelty of this work was to propose a model and its estimation at the master level by exploiting the sufficient statistics from the workers, in a DPM compliant approach. It takes advantage of the computing power of distributed systems by using parallel frameworks such as Spark [21]. As illustrated in figure 2, the DC-DPM solution distributes the Dirichlet Process by identifying local clusters on the workers and synchronizing these clusters on the master. These clusters are then communicated as a basis among workers for local clustering consistency. The Dirichlet Process is modified to consider this basis in each worker. By iterating this process the global consistency of DPM is sought in a distributed environment. The experiments of DC-DPM, using real and synthetic datasets, illustrate both the high efficiency and linear scalability of the approach. They report significant gains in response time, compared to centralized DPM approaches, with processing times of a few minutes, compared to several days in the centralized case. The

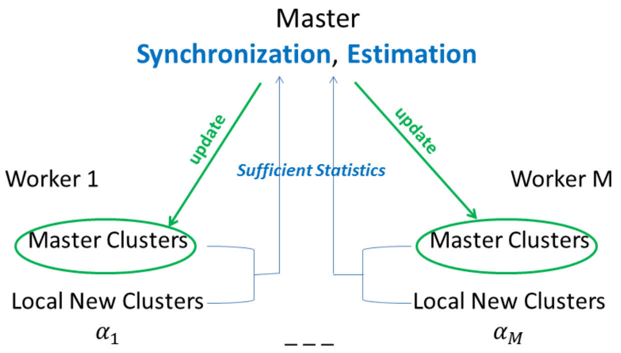

Figure 2: Diagram/workflow of the DC-DPM approach.

workflow of the DC-DPM approach is illustrated by Figure 2.

Our goal is to propose a parallel DPM approach for high dimensional data clustering based on the DC-DPM approach [16].

4 HD4C: High Dimensional Data Distributed Dirichlet Clustering

In this section, we present a novel parallel clustering approach called HD4C, adapted for high dimensional data and based on DC-DPM [16] described in section 3.

Actually, DC-DPM is a solution proposed to this issue when data is multivariate. This solution is based on a distributed DPM. In our case, the records are high dimensional data or signals (infinite dimension). In the case of infinite dimension, matrix computation is no more feasible (no inverse for example, no matrix product). In HD4C, we are not using the Lebesgue measure. We assume that observations are the addition of two gaussian processes : signal and noise. Therefore, we define a Radom-Nikodym derivative of gaussian measures [22] in order to compute a likelihood process.

A first attempt to work with this kind of data is to reduce their dimensionality, by sub-sampling the observations or projecting them into sub spaces like the one defined by a truncated basis of B-splines [23] or a truncated basis of kernel principal composant analysis [24]. Multivariate analysis, like SVM, k-means or DC-DPM, can then be applied.

A better approach is to continue working in infinite dimension to keep all information on the data. To compute a distributed DPM for high dimensional data or signals, we need to replace a matrix product by an inner product in an adequate space of functions and to find the adequate measure to compute the likelihood and the posterior. To do that, we used the properties of the Reproducible Kernel Hilbert Spaces (RKHS), as in [7].

RKHS (used for example in the Support Vector Machine approach) are very popular in machine learning thanks to “the representer theorem which simplified an infinite dimensional empirical risk minimization problem into a finite dimensional problem where the solution is included in the linear span of the kernel function evaluated at the training points” [25].

4.1 RKHS of Gaussian Process and DPM

We assume that the random variable of interest takes its values in a space of infinite dimension. Therefore, high dimensional data will be seen as trajectories of a random process Y: $Y=(Y(t))_{t\in[0,T]}$, where t stands for the general index of the Y function, t can be for example a time index in case of time series or a wavelength index in case of spectrum. In order to guarantee the existence of necessary conditional probabilities in the DPM algorithm, we will assume that the trajectories belong to the space of the integrable square functions ($L^2([0,T])$) on $[0,T]$ (from [26]). Our work focuses on Gaussian random process because “of its ability to avoid simple parametric assumptions and still build in a lot of structure”, [27]. In addition many calculations are facilitated in the Gaussian framework. For example, [28] stated that using Gaussian process for machine learning “turn out to be much more accurate than for parametric models of equal flexibility (such as multilayer perceptrons)”.

A Gaussian process $GP(m,K)$ is entirely defined by its mean function $m(t)$ and its covariance function $K(s,t)$, for all $t,s\in[0,T]$. The main idea behind the clustering with Gaussian Process is to use results from signal processing where the data is the sum of two Gaussian processes, namely a signal (a trajectory m_i issued from a $GP(m_0,K_0)$) and a noise (ε_i issued from a $GP(0,K)$):

$$Y_i=m_i+\varepsilon_i.$$

We assume that the signal is smoother than the noise in order to be able to detect it. To extract the signals and cluster them, we use the following DPM:

$$\begin{aligned} Y_i\,|\,m_i,K &\sim GP(m_i,K), i=1,\ldots,N\\ m_i &\sim G\\ G\,|\,m_0,K_0 &\sim DP(\alpha,GP(m_0,K_0)) \end{aligned}$$

DPM will create clusters of m_i where for all observations in cluster c, $m_i=\phi_c$. To run the DPM with algorithm 8 from Neal [16, 29], we need to define a posterior distribution $GP(m^*,K^*)$ for ϕ_c and the likelihood process $dGP(m_i,K)/dGP(0,K)$ for Y_i. From [28], the Reproducing Kernel Hilbert Space with reproducing kernel K, denoted H_K ”will turn out to contain expected values of m_i conditioned on a finite amount of information, thus the posterior mean function m^* we are interested in”.

Moreover, there exists a duality between a Gaussian process $GP(m,K)$ and H_K. H_K is a space of real functions defined on $[0,T]$ which verifies the following property: $\forall t\in[0,T]$, $\forall f\in H_K$, $f(t)=(f,K(.,t))_K$, where $(.,.)_K$ is the inner product of H_K. From [30], we define the random variable $(Y,f)_K$ like a stochastic integral. The properties of H_K allow to define the likelihood process [31, 32]:

$$(Y,K(,t))_K=Y(t) \quad (1)$$

$$f,g\in H_K \quad , \quad (f,g)_K=E[(Y,f)_K(Y,g)_K] \quad (2)$$

$$m_i\in H_K \quad , \quad \frac{dGP(m_i,K)}{dGP(0,K)}(Y_i)=e^{(Y_i,m_i)_K-\frac{1}{2}(m_i,m_i)_K} (3)$$

To ensure that $m_i\in H_K$, we must choose carefully the covariance function K_0, because the differentiability of m_i

up to a given order (and therefore the smoothness of m_i) can be controlled via the covariance function.

Finally, following [7, 33, 34], the posterior distribution for the signal of a cluster c is a Gaussian process, namely $\phi_c \,|\, (Y_i)_{c_i=c} \sim GP(m^*,K^*)$ with:

$$\begin{aligned} m^*(t) &= m_0(t)+(K_0(.,t),(\bar{Y}_c-m_0))_{K/n_c+K_0} \quad (4) \\ K^*(s,t) &= K_0(s,t)-(K_0(.,s),K_0(.,t))_{K/n_c+K_0} \quad (5) \end{aligned}$$

where the covariance functions K and K_0 are weakly continuous functions on $[0, T] \times [0, T]$; n_c and $\bar{Y}_c$ are respectively the number of observations and the mean function $\left(\bar{Y}_c=\frac{1}{n_c}\sum_{c_i=c} Y_i\right)$ in cluster c.

When K is non singular and weakly continuous, usual matrix approximations of the inner product results from [31]:

$$\begin{aligned} \lim_{L\to\infty} {}^t f^{(L)} {K^{(L)}}^{-1} g^{(L)} &= (f,g)_K \\ \lim_{L\to\infty} {}^t Y_i^{(L)} {K^{(L)}}^{-1} g^{(L)} &= (Y_i,g)_K \end{aligned}$$

where $(t^l)_{l=1\ldots L}$ is dense in $[0,\ T]$ and $f^{(L)} = (f(t^1), \ldots, f(t^L))$, $g^{(L)} = (g(t^1), \ldots, g(t^L))$ and $K^{(L)}$ is a $L \times L$ matrix whose elements are $K(t^l,t^j)$ for $1 \leq l,j \leq L$. Oya et al. [35] proposed a generalised numerical approach to estimate the inner product in H_k. In our approach (Section IV), we use a known analytical form for the inner product, which avoids matrix product or inversion and thus allows to escape the curse of dimensionality.

4.2 Massive Distribution and Spark

Clustering via Dirichlet Process Mixture based on Gibbs Sampling is unable to scale to large datasets due to its high computational costs associated with Bayesian inference. For this reason, we aim to implement a parallel algorithm for DPM clustering in a massively distributed environment called Spark which is a parallel programming framework aiming to efficiently process large datasets. This programming model can perform analytics with in-memory techniques to overcome disk bottlenecks. Similar to MapReduce [36], Spark can be deployed on the Hadoop Distributed File System (HDFS) [37]. Unlike traditional in-memory systems, the main feature of Spark is its distributed memory abstraction, called resilient distributed datasets (*RDD*), that is an efficient and fault-tolerant abstraction for distributing data in a cluster. With RDD, the data can be easily persisted in main memory as well as on the hard drive. Spark is designed to support the execution of iterative algorithms [21].

To execute a Spark job, we need a master node to coordinate job execution, and some worker nodes to execute a parallel operation. These parallel operations are summarized to two types: (i) Transformations: to create a new RDD from an existing one (*e.g.*, Map, MapToPair, MapPartition, FlatMap); and (ii) Actions: to return a final value to the user (*e.g.*, Reduce, Aggregate or Count) [21].

4.3 HD4C

Working in infinite dimension (functional data) allows to use information on the trajectories but also on their derivatives, which may reveal key information for the data clustering (see [38]). Indeed an Hilbert space (like the RKHS) is a space of integrable square functions ($L^2([0,T])$) on $[0,T]$, it is a special case of a Sobolev space. It means that a RKHS is a vector space of functions equipped with a norm that is a combination of L^p-norms of the function itself and its derivatives up to a given order. The given order is conditioned by the differentiability of the trajectories and therefore by the covariance function K of the random process Y.

In our experiments, we defined $Y_i \mid \theta_i = m_i, K$ as an autocorrelated Gaussian process called Ornstein-Uhlenbeck (OU) whose covariance function is defined as follows:

$$K(s,t) = \frac{\sigma^2}{2\beta} e^{-\beta|s-t|}, \quad (6)$$

where σ and β are two positive real.

Therefore, from [39], H_K is a space of differentiable functions in $[0,T]$ with the scalar product (defining the norm):

$$\begin{aligned} (f,g)_K = \frac{1}{\sigma^2} \int_0^T \Big(f'(t)g'(t) + \beta^2 f(t)g(t) \Big) dt \\ + \frac{\beta}{\sigma^2} \Big(f(0)g(0) + f(T)g(T) \Big). \end{aligned} \quad (7)$$

To ensure that $m_i \in H_K$, we used the prior $G = GP(m_0,K_0)$, where

$$K_0(s,t) = \frac{\sigma_0^2}{2\beta_0} e^{-\beta_0 (s-t)^2}.$$

This covariance gives very smooth trajectories (infinitely differentiable).

Other choice of covariance functions are possible for non smooth observations (like a Wiener process). Defining the covariance function K on the observations is equivalent to defining the kernel covariance K of the RKHS H_K. Defining a kernel K requires defining an inner product in H_K, which is equivalent to defining a metric, a distance between two observations $d(i,j) = (m_i - m_j, m_i - m_j)_K$. This led us to use a Sobolev metric for high dimensional Gaussian data (ie a distance between trajectories and their derivatives for OU Gaussian data) instead of the usual euclidean distance $\int_0^T (m_i(t) - m_j(t))^2 dt$, see the "changing metrics" discussion in [40].

Implementating this algorithm requires:

- The set of indexes used for computing the integrals in the inner product equation (7); for example in time series, it could be the observation time steps or not.
- An interpolation of the observations (if needed) to simplify the computation of the inner product. This interpolation can be used to adapt the observations to the covariance function K.
- Computation of the densities at the master and at the worker level, from equation (3). This requires estimating the hyperparameters β and σ. To avoid overly complex modelling, we have chosen to fix them empirically. As the Y_i curves are generated from Gaussian processes with covariance function K in (6), the parameters β and σ were determined from the empirical estimation of the intra-class variance-covariance matrix of the curves discretized in a few points.

We provide below more specific details:

Worker level

In the Gaussian process framework, the likelihood process is defined with respect to the Gaussian measure from $GP(0,K)$. Using [32] we have

$$F(y_i,\phi_c)=e^{(y_i,\phi_c)_K-\frac{1}{2}(\phi_c,\phi_c)_K}.$$

As the density of the predictive prior cannot be expressed with respect to the same Gaussian measure ($GP(0,K)$) than the likelihood, we approximated the integral in the MCMC algorithm, as suggested in algorithm 8 of [29], by drawing m realisations of ϕ_c.

To improve the variety of new candidate values of ϕ_c^{new}, we modified the original algorithm according to the following: $\phi_c^{new}(t)=m_0(t)+\zeta(t)$, where $\zeta(t)$ is a trajectory simulated from $GP(m_0,K)$ and $m_0(t)$ is randomly simulated from a truncated polynomial basis (the basis order is also randomly chosen).

Following [41], we used an inverse Gamma prior to infer the parameter α_j.

The following algorithm 1 summarizes the worker level.

Algorithm 1 DPM at worker j

```
for each data $y_i$ do
    Draw $m$ values $\phi_c^{new}$
    Draw individual cluster label $c_i$ in addition to existing $\phi_1,...,\phi_C$
    $P(c_i=c|\{c_l\}_{l\neq i},y_i,\{\phi\},\{w\},\alpha_j)\propto$
    $\begin{cases} \frac{\#(c)+\alpha_j w_c}{N_j-1+\alpha_j}e^{(y_i,\phi_c)_K-\frac{1}{2}(\phi_c,\phi_c)_K}, c=1,...,C \\ \frac{1}{m}\frac{\alpha_j w_u}{N_j-1+\alpha_j}e^{(y_i,\phi_c^{new})_K-\frac{1}{2}(\phi_c^{new},\phi_c^{new})_K}, c=1,...,m \end{cases}$
end for
Update of $\alpha_j$
```

where the weight w_c is the proportion of observations from cluster c evaluated on the whole dataset and w_u the proportion of non affected observations (awaiting the creation, innovation, discover of their real clusters), with $w_u+\sum_{c=1}^{C}w_c=1$. Therefore, these parameters are updated at the master level during the synchronization.

Master level

Instead of drawing new values ϕ_c^{new}, the proposed algorithm reuses the center values of the clusters received from the workers, namely $\phi_k^{worker_j}$.

The approximation of ϕ is updated by computing the posterior mean in each cluster, equation (4), to which we add a noise drawn from a $GP(0,K/n_c)$.

Following [16], we use a Dirichlet prior to infer $(w_1,...,w_K,w_u)$.

The master lever is outlined in algorithm 2.

5 Experiments

The parallel experimental evaluation was conducted on a computing cluster of 32 machines, each operated by Linux, with 64 Gigabytes of main memory, Intel Xeon CPU with 8 cores and 250 Gigabytes hard disk. We compared our approach to K-means, which is one of the most commonly used clustering algorithms. We used an implementation available at Spark's machine learning library (MLlib) [42].

Algorithm 2 DPM at master level

```
for each cluster $k$ from worker $j$ do
    Draw cluster label $z_{j,k}$ from
    $P(z_{j,k}=c|\{c\}_{\neq j,k},\bar{y}_{j,k},\{\phi\},\gamma)\propto$
    $\begin{cases} \frac{\#(c)}{N-1+\gamma}e^{(\bar{y}_{j,k},\phi_c)_K-\frac{1}{2}(\phi_c,\phi_c)_K}, c=1,...,C \\ \frac{\gamma}{N-1+\gamma}e^{(\bar{y}_{j,k},\phi_k^{\mathrm{worker}j})_K-\frac{1}{2}(\phi_k^{\mathrm{worker}j},\phi_k^{\mathrm{worker}j})_K} \end{cases}$
end for
Update of $\phi$ and $(w_1,...,w_K,w_u)$
```

The first step of HD4C is a distributed K-means that sets the initial state (usually we set K to be one tenth of the dataset size).

Reproducibility : All our experiments are fully reproducible. We make our code and data available at `https://github.com/khadidjaM/HD4C`.

In the rest of this section, we describe the datasets in Section 5.1 and our evaluation criteria in Section 5.2. Then, in Section 5.3, we measure the performances, in response time, of our approach by reporting its scalability and speed-up. We evaluate the clusters obtained by HD4C in the case of real and synthetic dataset in Section 5.4.

5.1 Datasets

We carried out our experiments on two real world datasets and many synthetic datasets.

Our synthetic data was generated using a two-steps principle. First we generated four cluster centers according to the following polynomials :

$$\begin{cases} s_1(t)= & 0.11t^3-0.16t^2+0.55t \\ s_2(t)= & -0.75t^4+1.49t^3-0.91t^2+0.17t \\ s_3(t)= & 3.91t^5-9.77t^4+0.854t^3-3.05t^2+0.37t \\ s_4(t)= & -20.09t^6+60.26t^5-68.22t^4+36t^3 \\ & -8.71t^2+0.76t \end{cases}$$

In the second step, we generated the data corresponding to each center, by using a Gaussian process of mean s_i and a covariance given by an Ornstein-Uhlenbeckh process parametrized by $\beta=10$ and $\sigma=2.5$. We independently generated a batch of 5 datasets having size 200K, 400, 600, 800K and 1M time series of 100 points, the latter dataset is about 2 Gigabytes. Figures 3 and 4 give a visual representation of our synthetic dataset. Each cluster is assigned a color and represented by 10 time series. This type of generator is widely used in statistics, where methods are first evaluated on synthetic data before being applied on real data.

The first real world dataset corresponds to more than five thousands accelerometer time series which have been measured by sensor on 13 sheep (as in figure 1). Each time series is made of 500 observation times and has been visually assigned to one of six activities (STANDING-GRAZING, STANDING-

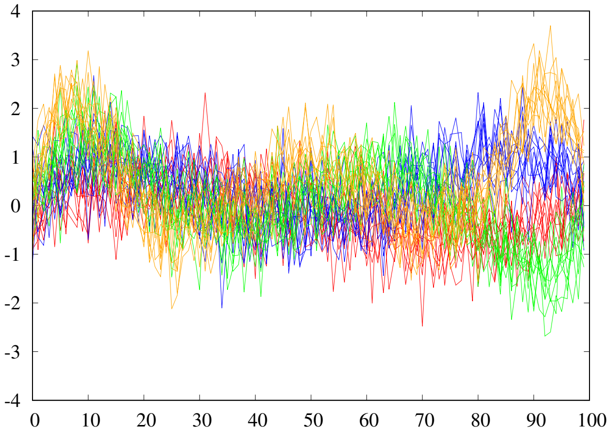

Figure 3: Visual representation of the synthetic dataset clusters.

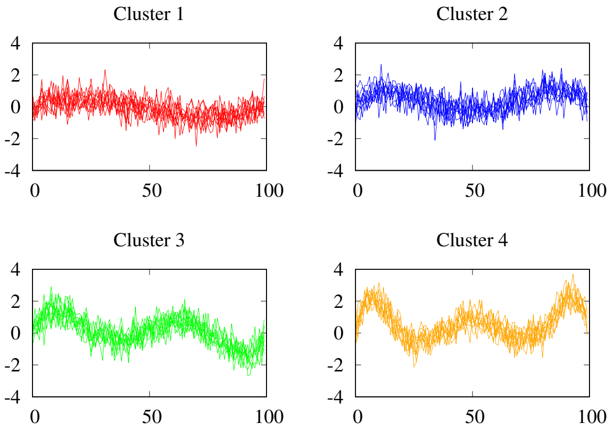

Figure 4: Visual representation of the synthetic dataset with separated clusters.

EATING BRUSH, STANDING-RUMINATING, WALKING, RUNNING, STANDING-IMMOBILE). Accelerometers captured 3-axial acceleration at a constant rate of 100Hz. The sensor signals were pre-processed and for each activity of interest, sampled in fixed-width of 5 seconds (500 values / a time series). Each of the three axial acceleration gives a different information for the zoologist, so HD4C clustering was performed by axis (horizontals (x and y) and vertical (z)). The objective was to discover the underlying structures of each axis and then to link these structures to sheep activities. Figures 5 and 6 represent one axis of the accelerometers dataset. Each label of activity is assigned a color and represented by 5 time series.

The second real dataset corresponds to more than 4K spectrum of 680 dimensions representing a protein rate measured on 10 different products: rapeseed (CLZ), corn gluten (CNG), sun flower seed (SFG), grass silage (EHH), full fat soya (FFS), wheat (FRG), sun flower seed (SFG), animal feed (ANF), soya meal (TTS), maïs (PEE), milk powder and whey (MPW). Figure 7 gives a visual representation of the spectral data. Each product is assigned a color and represented by 50 spectrums.

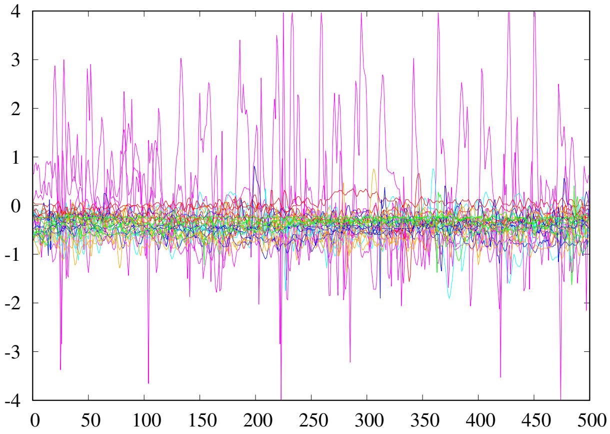

Figure 5: One axis visual representation of labeled accelerometers data

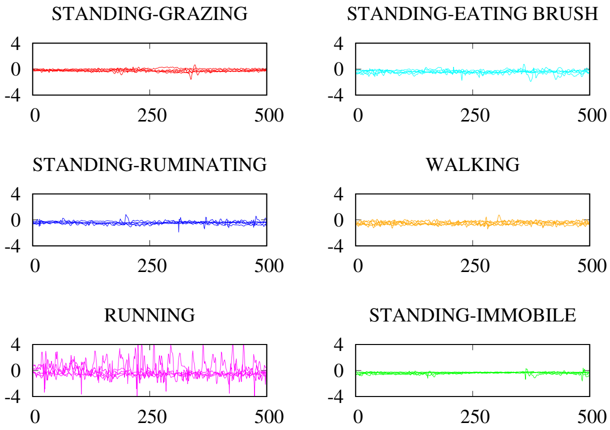

Figure 6: Separated clusters of one axis accelerometers data

5.2 Clustering Evaluation Criteria

There are two cases for evaluating the results of a clustering algorithm. Either there is a ground truth available, or there is not. In the case of an available ground truth, there are measures allowing to compare the clustering results to the reference, such as the Adjusted Rand Index (ARI): it is the corrected-for-chance version of the Rand Index [43], which is a function that measures the similarity between two data clustering results, for example between the ground truth class assignments (if known) and the clustering algorithm assignments. ARI values are in the range [-1,1] with a best value of 1. It is usually exploited for experiments when one wants to check performances in a controlled environment,

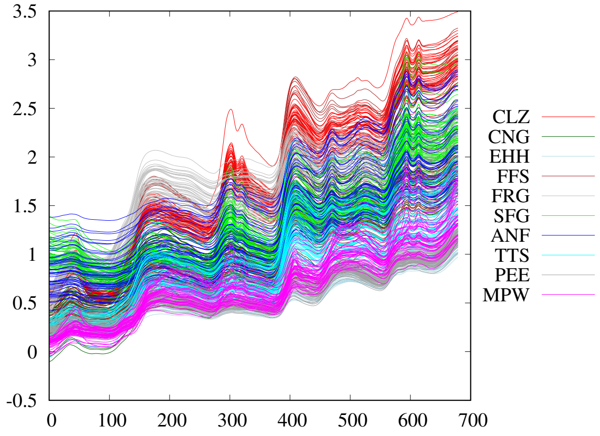

Figure 7: Visual representation of the spectral dataset

on synthetic data or labelled real data.

In the case where there is no ground-truth (which is the usual case, because we don't know what should be discovered in real world applications of a clustering algorithm) the validation of the results is not straightforward. In our experiment we have checked K, the number of discovered clusters versus the expected number of clusters according to expert.

5.3 Response Time

In this section we measure the clustering time in HD4C. Figure 8 reports the response times on our synthetic data, HD4C is run on a computing cluster of 16 nodes. The clustering time increases with the number of data, our approach benefits from linear scalability with the dataset size. For a dataset of 200K data points, HD4C performs the clustering in about 12 minutes, while a centralized approach does not scale and cannot execute on such dataset size, it needs several days on a single machine.

Figures 9, 10 and 11 illustrate the parallel speed-up of our approach on 200K time series from the synthetic dataset, on accelerometers data from the first real world dataset, and on spectrums from the second real dataset. These experiments are conducted on 4, 8 and 16 nodes which correspond to 32, 64 and 128 workers (each node has 8 cores). The results show optimal or near optimal gain. On the accelerometers dataset there is not a big difference between 8 and 16 nodes because this dataset is not big, and distributing it on 8 or 16 nodes is super fast at workers level while the synchronisation at the master level takes almost the same time. Another reason is that the computing nodes do not have the same performances, those who have finished must wait for the slower nodes.

5.4 Clustering Evaluation

In the following experiments, we evaluate the clustering performance of HD4C and compare it to the K-means approach.

Table 1 reports the ARI values computed between the clustering obtained and the ground truth, the estimated values of parameters $\hat{\sigma}$ and $\hat{\beta}$, and the number of clusters, obtained with HD4C on our synthetic data while increasing the dataset size. The HD4C is run on a cluster of 16 nodes. HD4C performs well, the ARI values are almost equal to 1 (best value), the number of discovered clusters is equal to the real number of clusters, the estimated values of $\hat{\sigma}$ and $\hat{\beta}$ are close to the parameters used for simulating the data. Note also that the estimated ratio $\frac{\hat{\sigma}^2}{2\hat{\beta}}$ converges to the true simulated ratio $\frac{\sigma^2}{2\beta}$, which corresponds to the variance on the diagonal of K in (6).

Figure 12 reports the Adjusted Rand Index values (described in section 5.2) obtained by performing K-means approach on 200K time series from the synthetic dataset as a function of the number of clusters, it is run on two nodes (16 workers). The K-means approach does not reach the best value 1, the peak of these values is 0.90 but with 9 clusters which is not the real number in the ground truth, while with the real number of clusters (4 clusters) the ARI value is 0.79.

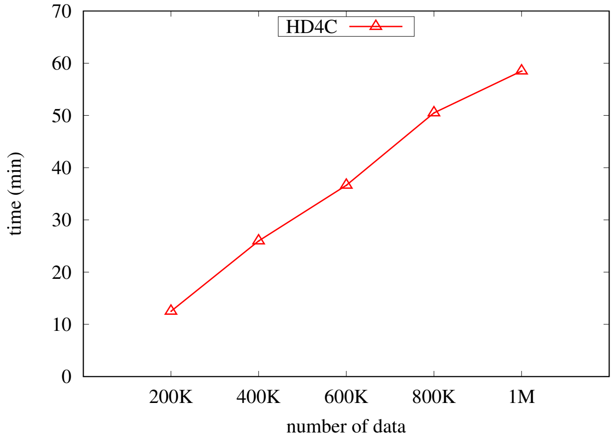

Figure 8: Response time (minutes) of HD4C as a function of the dataset size.

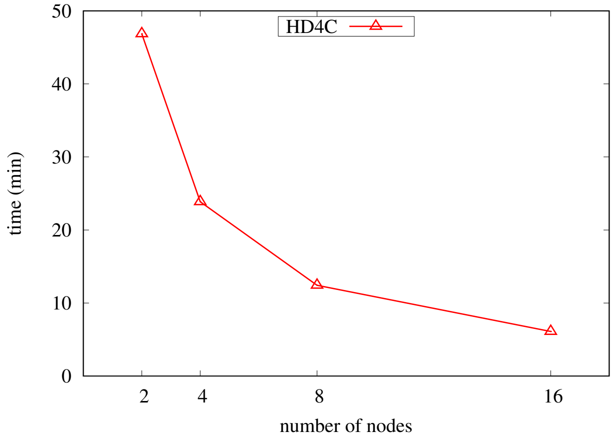

Figure 9: Clustering time as a function of the number of computing nodes on the synthetic data.

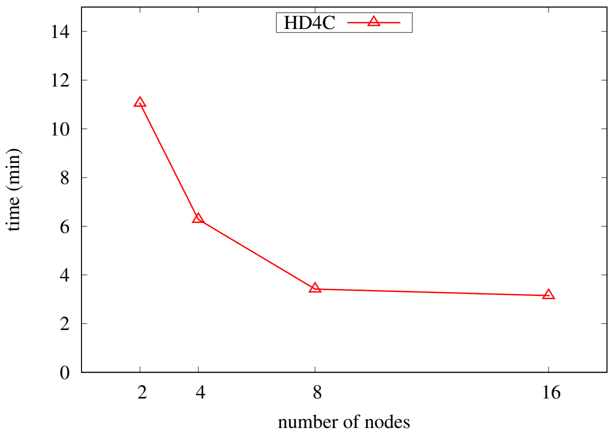

Figure 10: Clustering time as a function of the number of computing nodes on the accelrometers data.

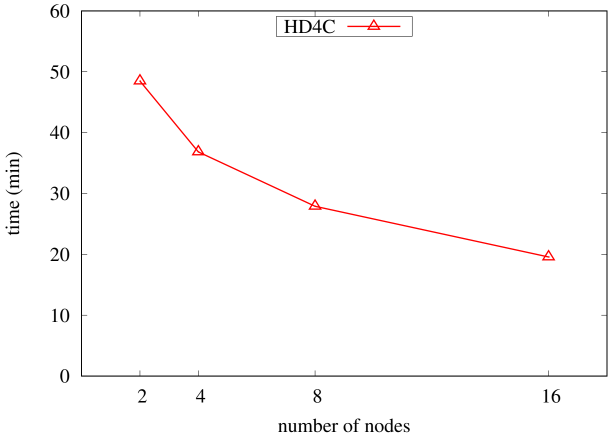

Figure 11: Clustering time as a function of the number of computing nodes on the spectral data.

Table 1: Clustering evaluation criteria obtained with HD4C (synthetic data).

	ARI	$\hat{\sigma}$	$\hat{\beta}$	$\hat{\sigma}^2/2\hat{\beta}$	**Clusters**
200K	1.00	2.57	10.59	0.31	4
400K	1.00	2.13	7.25	0.31	4
600K	0.99	2.15	7.44	0.31	4
800K	1.00	2.28	8.30	0.31	4
1M	0.99	2.13	7.25	0.31	4

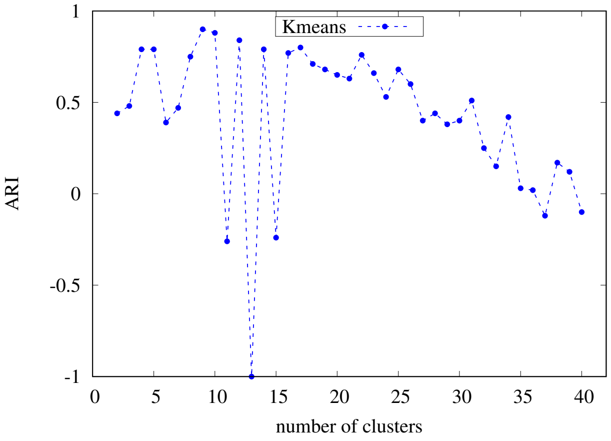

Figure 12: ARI values of K-means as a function of the number of clusters.

K-means suffers from the convergence to a local minimum which may produce ”wrong” results, as illustrated for example in Table 2. This table shows the results of K-means performed on 600K time series of the synthetic dataset with the right number of clusters (4 clusters, each containing 150K data) and run on 16 nodes. Each line of Table 2 represents one cluster obtained by K-means and reports the number of data obtained in each cluster: the cluster 2 obtained by K-means regroups the two real clusters 1 and 3, while the real cluster 2 is divided between clusters 1 and 3 discovered by K-means.

Table 2: Example of K-means convergence to a local minimum.

	Ground truth			
	1	2	3	4
1	0	75945	0	0
2	150000	0	150000	0
3	0	74055	0	0
4	0	0	0	150000

By comparison, when applying HD4C on the same dataset, the right number of clusters is discovered and all the data except a few ones are affected to the true clusters, as presented in Table 3.

Repeating the clustering on accelerometers data many times by HD4C and K-means, we obtained the ARI values showed on table 4. Our approach performs better than the K-means approach, the average value obtained by HD4C is 0.50 which is a good value regarding the shapes of data in clusters: STANDING-GRAZING, STANDING-EATING BRUSH, STANDING-RUMINATING, WALKING. The true labels have been visually assigned by the experts, by observing the three axes at the same time. It is difficult to label them by only analysing one axis at a time (see figure 6).

Table 3: Number of data obtained by HD4C in each cluster compared to the ground truth.

	Ground truth			
	1	2	3	4
1	0	149989	0	0
2	150000	0	1	0
3	0	11	149999	0
4	0	0	0	150000

HD4C is not intended to cluster multidimensional time series.

Table 4 also represents the ARI values obtained with the real world datasets both for HD4C and K-means. K-means was processed with the number of clusters found by HD4C. Each time we repeat the HD4C clustering we find a number close to the number of labels given by the experts.

Table 4: Clustering evaluation criteria obtained with HD4C and K-means on real datasets.

	HD4C		**K-means**
	ARI	**Clusters**	**ARI**
Accelerometers	0.50	8	0.11
Spectrums	0.34	9	0.32

6 Conclusion

We proposed HD4C, a novel and efficient parallel solution to perform clustering via DPM on large amount of infinite dimensional data. These infinite dimensional data include lengthy time series or spectral data for example. We evaluated the performance of our solution over real world and synthetic datasets. The experimental results illustrate the high performance of HD4C with results that are comparable to K-means, one of the most commonly used clustering algorithms. Overall, the experimental results show that by using our parallel techniques, the clustering of very large volumes of data can now be done in small execution times, which is impossible to achieve using a centralized DPM approach. A nice perspective of our approach is now to extend the HD4C algorithm to multivariate functional data, like the accelerometer dataset which contains time series recorded according to three axes.

7 Acknowledgements

The research leading to these results has received funds from the European Union's Horizon 2020 Framework Programme for Research and Innovation, under grant agreement No. 732051.

This work was also supported by the 'Infrastructure Biologie Santé' PHENOME-EMPHASIS project (ANR-11-INBS-0012) funded by the National Research Agency and the 'Programme d'Investissements d'Avenir' (PIA).

The accelerometer data come from CLOChèTE project, supported by CASDAR funds.

The spectrum dataset comes from V. Baeten's team at the Walloon Agricultural Research Centre (CRA-W).

References

[1] A. Alamsyah and B. Nurriz, "Monte carlo simulation and clustering for customer segmentation in business organization," in *2017 3rd International Conference on Science and Technology - Computer (ICST)*, July 2017, pp. 104–109.

[2] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, Oct 2004. [Online]. Available: https://doi.org/10.1023/B:AIRE.0000045502.10941.a9

[3] Ordovás-Pascual, I. and Sánchez Almeida, J., "A fast version of the k-means classification algorithm for astronomical applications," *Astronomy & Astrophysics*, vol. 565, p. A53, 2014.

[4] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (methodological)*, pp. 1–38, 1977.

[6] M. D. Escobar, "Estimating normal means with a dirichlet process prior," *Journal of the American Statistical Association*, vol. 89, no. 425, pp. 268–277, 1994.

[7] D. Juery, C. Abraham, and B. Fontez, "Classification bayésienne non supervisée de données fonctionnelles," *Journal de la Société Française de Statistique*, vol. 155, no. 2, pp. 185–201, 2014.

[8] S. Prasad and L. M. Bruce, "Limitations of principal components analysis for hyperspectral target recognition," *IEEE Geoscience and Remote Sensing Letters*, vol. 5, no. 4, pp. 625–629, 2008.

[9] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering–a decade review," *Information Systems*, vol. 53, pp. 16–38, 2015.

[10] E. Keogh and J. Lin, "Clustering of time-series subsequences is meaningless: implications for previous and future research," *Knowledge and Information Systems*, vol. 8, no. 2, pp. 154–177, 2005.

[11] M. L. L. García, R. García-Ródenas, and A. G. Gómez, "K-means algorithms for functional data," *Neurocomputing*, vol. 151, pp. 231–245, 2015.

[12] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing sax: a novel symbolic representation of time series," *Data Mining and Knowledge Discovery*, vol. 15, no. 2, pp. 107–144, 2007.

[13] S. Soheily-Khah, A. Douzal-Chouakria, and E. Gaussier, "Generalized k-means-based clustering for temporal data under weighted and kernel time warp," *Pattern Recognition Letters*, vol. 75, pp. 63–69, 2016.

[14] Y.-H. Cheng, T.-M. Huang, and S.-F. Yang, "A clustering method for misaligned curves," *arXiv preprint arXiv:1801.00382*, 2018.

[15] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[16] K. Meguelati, B. Fontez, N. Hilgert, and F. Masseglia, "Dirichlet process mixture models made scalable and effective by means of massive distribution," in *SAC: Symposium on Applied Computing*, Limassol, Cyprus, Apr. 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01999453

[17] D. Lovell, R. P. Adams, and V. Mansingka, "Parallel markov chain monte carlo for dirichlet process mixtures," in *Workshop on Big Learning, NIPS*, 2012.

[18] S. Williamson, A. Dubey, and E. Xing, "Parallel markov chain monte carlo for nonparametric mixture models," in *International Conference on Machine Learning*, 2013, pp. 98–106.

[19] R. Wang and D. Lin, "Scalable estimation of dirichlet process mixture models on distributed data," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, ser. IJCAI'17. AAAI Press, 2017, pp. 4632–4639. [Online]. Available: http://dl.acm.org/citation.cfm?id=3171837.3171935

[20] Q. Zhu, G. Batista, T. Rakthanmanon, and E. Keogh, "A novel approximation to dynamic time warping allows anytime clustering of massive time series datasets," in *Proceedings of the 2012 SIAM international conference on data mining*. SIAM, 2012, pp. 999–1010.

[21] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *HotCloud*, 2010.

[22] L. A. Shepp, "Radon-nikodym derivatives of gaussian measures," *Ann. Math. Statist.*, vol. 37, no. 2, pp. 321–354, 04 1966. [Online]. Available: https://doi.org/10.1214/aoms/1177699516

[23] C. Abraham, P. A. Cornillon, E. Matzner-Løber, and N. Molinari, "Unsupervised curve clustering using b-splines," *Scandinavian Journal of Statistics*, vol. 30, no. 3, pp. 581–595, 2003. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-9469.00350

[24] M. Fauvel, J. Chanussot, and J. A. Benediktsson, "Kernel principal component analysis for the classification of hyperspectral remote sensing data over urban areas," *EURASIP J. Adv. Signal Process*, vol. 2009, pp. 11:1–11:14, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1155/2009/783194

[25] K. Ø. Mikalsen, F. M. Bianchi, C. Soguero-Ruiz, and R. Jenssen, "Time series cluster kernel for learning similarities between multivariate time series with missing data," *Pattern Recognition*, vol. 76, pp. 569–581, 2018.

[26] R. M. Dudley, "Real analysis and probability. wadsworth & brooks," *Cole, Pacific Groves, California*, 1989.

[27] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. Massachusetts Institute of Technology, 2006.

[28] M. Seeger, "Gaussian processes for machine learning," *International Journal of Neural Systems*, vol. 14, no. 02, pp. 69–106, 2004, pMID: 15112367. [Online]. Available: https://doi.org/10.1142/S0129065704001899

[29] R. M. Neal, "Markov chain sampling methods for dirichlet process mixture models," *Journal of Computational and Graphical Statistics*, vol. 9, no. 2, pp. 249–265, 2000.

[30] E. Parzen, "Regression analysis of continuous parameter time series," in *Int. ISPASS*, 2010.

[31] ——, "Statistical inference on time series by hilbert space methods, i," Stanford Univ CA Applied Mathematics and Statistics Labs, Tech. Rep., 1959.

[32] ——, "Probability density functionals and reproducing kernel hilbert spaces," in *Proceedings of the Symposium on Time Series Analysis*, vol. 196. Wiley, New York, 1963, pp. 155–169.

[33] M. F. Driscoll, "The signal-noise problem—a solution for the case that signal and noise are gaussian and independent," *Journal of Applied Probability*, vol. 12, no. 1, pp. 183–187, 1975.

[34] A. W. van der Vaart, J. H. van Zanten *et al.*, "Reproducing kernel hilbert spaces of gaussian priors," in *Pushing the limits of contemporary statistics: contributions in honor of Jayanta K. Ghosh*. Institute of Mathematical Statistics, 2008, pp. 200–222.

[35] A. Oya, J. Navarro-Moreno, and J. C. Ruiz-Molina, "Numerical evaluation of reproducing kernel hilbert space inner products," *IEEE Transactions on Signal Processing*, vol. 57, no. 3, pp. 1227–1233, 2009.

[36] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[37] J. Shafer, S. Rixner, and A. L. Cox, "The hadoop distributed filesystem: Balancing portability and performance," in *Int. ISPASS*, 2010.

[38] N. Coffey and J. Hinde, "Analyzing time-course microarray data using functional data analysis-a review," *Statistical Applications in Genetics and Molecular Biology*, vol. 10, no. 1, 2011.

[39] A. Berlinet and C. Thomas-Agnan, *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011.

[40] G. Saporta, "Data analysis for numerical and categorical individual time-series," *Applied Stochastic Models and Data Analysis*, vol. 1, no. 2, pp. 109–119, 1985. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/asm.3150010204

[41] M. D. Escobar and M. West, "Bayesian density estimation and inference using mixtures," *Journal of the American Statistical Association*, vol. 90, no. 430, pp. 577–588, 1995.

[42] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[43] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *J. Mach. Learn. Res.*, vol. 11, pp. 2837–2854, Dec. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1756006.1953024