Incremental Learning of Probabilistic Models in High-Dimensional Space

Jaime Ferreira
Instituto Superior Técnico - Universidade de Lisboa
jaime.ferreira@l2f.inesc-id.pt

***Abstract*—Modeling data is central to all machine learning tasks. Many approaches operate in batch mode, and require data to be fully available a priori. However, on some domains, we may want to continuously build and update a model as more data arrives. On big data domains the data may not even fit entirely in memory, making batch approaches inadequate. Online approaches are more suitable for these tasks as they do not require data to be available a priori. Some go further, and do not need to store all previously observed samples. In this thesis, we describe and implement a state of the art online kernel density estimation approach, that allows to continuously model data points by building and maintaining a Gaussian mixture model of the data. Our implementation, which we called xokde++, is also designed to handle high dimensionality and arbitrary data distributions, such as degenerate and skewed data. Furthermore the code is extensible and easy to integrate in larger projects. Comparing to the current state of the art, our implementation is up to 40 times faster, needs 90% less memory, has greater numerical robustness, and produces, on average, models with higher quality. To show extensibility and further improve numerical stability we also implement a variant of xokde++ that replaces the full covariance Gaussian kernels by diagonal covariance ones, with positive results on both numerical robustness and computational performance.**

I. Introduction

The ability to model data is central to all machine learning tasks. In most tasks. data is available beforehand as a training set, and performance is evaluated in a separate test set. However, sometimes such as in big data applications, data does not fit in memory or is not completely available beforehand. In a more extreme scenarios the data may change over time, and thus is desirable to build and maintain a model of current and past data, but capable of some degree of forgetfulness. This is critical in long running applications where we want continuously receive and model data for months or years.

Keeping all observed data samples and retraining the whole model at each update, or even periodically is computationally unfeasible and wasteful. A better approach is to keep a simplified model of the data where similar data samples are grouped together and represented by a single entity. Gaussian mixture models (GMM) are especially useful for this task as they provide a generative model of data. Grouping together several observations under a single Gaussian component avoids having to keep all previously observed samples, needing only to keep the Gaussian component parameters. One approach to train GMMs is to use the Expectation Maximization algorithm [1] (EM). However, this approach suffers two caveats. First the number of Gaussian components must be defined a priori, which limits applicability in long term modeling. Plus, it suffers from initialization issues where an improper choice may lead to poor models unable to capture the complete structure of the underlying pdf. This is typically solved by using training data to find the best initialization [2]. However, in some scenarios no training set is available. And even if a training set is available, data may change over time, making the initial initialization, optimal for the original data, sub optimal for the current data. Finally, although incremental versions of EM exist [3], they still need to keep parts of the previously observed data and to define, a priori, the target number of components.

We define our setup to be one where the data to be modeled is not known a priori, and where the shape of the target distribution changes over time. Since we assume no training information is available in our scenario, we avoid assuming anything about the data nature, feature length, and underlying distribution. We also want to continuously update the model in long operation times while keeping the current model updated and ready to use. Since memory is finite, having a machine permanently capturing and processing data samples makes saving all previously observed samples a computationally unfeasible approach. Furthermore, since the goal is to model data through months or years, we want our model to be able to evolve as new data appears while making older data, possibly obsolete, progressively less important.

Kernel density estimation methods are nonparametric approaches that do not require any prior definition of the number of components, thus avoiding the initialization problem of EM. The drawback is that each observation is represented as single component in the mixture. This implies that the complexity grows linearly with the number of observed data. A solution for this is to compress the model either to a predefined number of components [4] or to optimize some data-driven choice [5]. Ozertem et al. [6] have a different approach in which the model compression is viewed as a clustering problem. Adapting kernel density estimation methods to online operation instead of batch is a non trivial task. The main difficulty lies in maintaining sufficient information of the estimated models to properly generalize to unobserved data and to adjust model complexity and parameters without directly accessing all previously observed samples. Han et al. [7] proposed an online approach based on mean-shift mode finding, and in which each new sample is added to the model as a Gaussian component. However this approach is sensitive to non-Gaussian areas due to skewed or heavy tailed data. Declercq and Piater [8] propose a two level approach where the key idea is that

each component of the (non-overfitting) mixture is in turn represented by an underlying mixture that represents data very precisely (possibly overfitting). This allows the model to be refined without sacrificing accuracy, as a merge in the upper level can later be accurately split. They also tackle the problem of non-Gaussian data by using a uniform distribution to represent regions where data is non-Gaussian.

Kristan et al [9], [10] recently proposed a nonparametric approach called *online kernel density estimator* (oKDE). This approach does not attempt to build a model of the target distribution directly. Rather, it maintains a nonparametric model of the data itself as mixture of Gaussian and Dirac-delta functions called *sample distribution*. This model is then used to compute the kernel density estimate of the target distribution. This separation is particularly useful in our setup, where the target distribution may change over time. Furthermore, being online, the model can be updated in single sample updates, as each new sample is simply a Dirac-delta function. To keep complexity low, the model is compressed from time to time using hierarchical clustering, which approximates clusters of components by single Gaussians. This approach allows to model multivariate data while making little assumptions on data nature. More importantly, it contains a forgetting factor, where old information progressively loses weight in the model. This allows to continuously model and adapt to new information even where the target distribution changes over time. Furthermore, this approach does not need to keep all previously observed samples, as it compresses similar observations under a single Gaussian. However if the target distribution changes sufficiently this merges may be no longer valid. To retain ability to later split these compressions, each component has an underlying model that provides sufficient information for a split. All these properties make this an appealing approach for a long running application with high dimensional sets of features such as face recognition.

As stated, we want to be able to keep and maintain a model learning in permanent operation. However, we also want to keep it permanently available to use with the most up to date state possible. This means that there is a need for computational speed component in our requirement. Unfortunately, the existing code for the oKDE implementation is MATLAB research code and is not optimized for speed. Furthermore, it is also not designed to handle very high dimensionality such as 1000 or 5000 dimensions, failing to fulfill the previous requirement of handling arbitrary feature length and structure.

Our work closely follows the approach described by Kristan et al [9], [10] and consists in an equivalent but faster and more robust version of the original oKDE. Implemented in C++, an efficient object oriented programing language, it also aims at flexibility and extensibility, thus easing the effort in exploring alternative lines of work in search of further improvements to the online estimation of kernel densities. The goal is to produce a fast, responsive and accurate, generic data modeling tool that is sufficiently robust to handle permanent operation through very long periods of time.

The following sections start by describing the original online kernel density estimation (Section II) method, we then describe our contributions to the original approach (Section III). The evaluation setup (Section IV) is presented next, followed by results (Section V), before concluding (Section VI).

II. Online Kernel Density Estimation

The state of the art work on online kernel density estimation presented before, was not specifically developed with high dimensionality in mind. Since the model is based on the multivariate Gaussian distribution then, by definition, it allows an arbitrary number of dimensions. But computational complexity and memory requirement scale quadratically with the number of dimensions, which makes this approach computationally prohibitive very fast. Plus, even if time and memory resources are available, numerical problems may arise due to overflows, underflows and precision issues. The original authors provide a MATLAB implementation, but it suffers from memory and computational inefficiencies, which are exacerbated as the number of dimensions grow, and would severely limit the dimensionality we would be able to work. Due to this, we fully reimplemented the oKDE approach [9] in C++. In the process, we also took into consideration the high dimensionality setup by avoiding, where possible, numerical instability prone operations. In the next subsections we will describe the theoretical background of the implemented approach, and implementation bound decisions are described where appropriate.

A. Model Definition

We define the sample model as a d-dimensional, N-component Gaussian mixture model.

$$p_s(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi_{\Sigma_i}(\mathbf{x} - \mathbf{x}_i) \tag{1}$$

where $\mathbf{x}$ is a D-dimensional data vector (i.e., the observation features), w_i are the mixture weights, and $g(\mathbf{x}|\mu_i, \Sigma_i)$, are the component Gaussian pdfs. Each component is a D-variate Gaussian probability density function of the form,

$$\phi_{\Sigma_i}(\mathbf{x}-\boldsymbol{\mu_i}) = \frac{1}{2\pi^{D/2}|\Sigma_i|^{1/2}} \exp\left\{\frac{1}{2}(\mathbf{x}-\mu_i)'\Sigma_i^{-1}(\mathbf{x}-\mu_i)\right\}, \tag{2}$$

with mean vector μ_i and covariance matrix Σ_i. Also, the sum of the mixture weights is constrained such that $\Sigma_{i=1}^{N} w_i = 1$.

To maintain a low complexity during online operation, we compress the sample distribution when certain criteria is met. However, as stated before, a valid compression now, may later prove to be invalid. To detect and recover from these over-compressions we keep an additional model of data for each component of the mixture. Our model of the *observed samples* is therefore defined as:

$$\mathbf{S}_{\text{model}} = \{p_s(\mathbf{x}), \{q_i(\mathbf{x})\}_{i=1:N}\} \tag{3}$$

where $p_s(\mathbf{x})$ is the sample distribution and $q_i(\mathbf{x})$ is a mixture model (with at most two components) for the i-th component in $p_s(\mathbf{x})$.

The kernel density estimate (KDE) is defined as a convolution of $p_s(\mathbf{x})$ by a kernel with a covariance matrix (bandwidth) $\mathbf{H}$:

$$\hat{p}_{\text{KDE}}(\mathbf{x}) = \phi_{\mathbf{H}}(\mathbf{x}) * p_s(\mathbf{x}) = \sum_{i=1}^{M} w_i \phi_{\mathbf{H}+\Sigma_i}(\mathbf{x} - \mathbf{x}_i) \tag{4}$$

B. Optimal Bandwidth

The goal of all KDE methods is to determine the kernel bandwidth H such that the distance between the $\hat{p}_{\text{KDE}}(\mathbf{x})$ and the unknown pdf $p(\mathbf{x})$ that generated the data, is minimized. If we assume that the structure of $\mathbf{F}$ is known and rewrite the bandwidth matrix in terms of scale β and structure $\mathbf{F}$, we have

$$\mathbf{H} = \beta^2 \mathbf{F} \tag{5}$$

such that the finding the optimal bandwidth is equivalent to finding the optimal scaling

$$\beta_{\text{opt}} = [d(4\pi)^{d/2} N R(p, F)]^{-1/(d+4)} \tag{6}$$

where the term $R(p, F)$ can be approximated by $\hat{R}(p, F, \mathbf{G})$

$$\hat{R}(p, F, \mathbf{G}) = \sum_{i=1}^{N} \sum_{j=1}^{N} w_i w_j \phi_{\mathbf{A}_{ij}^{-1}}(\Delta_{ij}) \times \left[[2\text{tr}(F^2 \mathbf{A}_{ij}^2)][1 - 2m_{ij}] + \text{tr}^2(F\mathbf{A}_{ij})[1 - m_{ij}]^2 \right] \tag{7}$$

For reading simplicity, we used the following definitions:

$$\mathbf{A}_{ij} = (\Sigma_{gj} + \Sigma_{sj})^{-1}, \quad \Delta_{ij} = \mu_i - \mu_j, \quad m_{ij} = \Delta_{ij}^T \mathbf{A}_{ij} \Delta_{ij} \tag{8}$$

In addition, Σ_{gj} is defined as

$$\Sigma_{gj} = \mathbf{G} + \Sigma_{sj} \tag{9}$$

where Σ_{sj} is the covariance of the j'th component of $p_s(\mathbf{x})$. The term $\mathbf{G}$ is the *pilot bandwidth*, which is estimated by a multivariate normal-scale rule of the distributions derivative

$$\mathbf{G} = \hat{\Sigma}_{smp} \left(\frac{4}{(d+2)N} \right)^{2/(d+4)} \tag{10}$$

where $\hat{\Sigma}_{smp}$ is the covariance of the approximation of the entire model by a single Gaussian.

Resorting to the practical assumption that the structure of the bandwidth $\mathbf{H}$ can be reasonably well approximated by the structure of the covariance matrix of the observed samples [11], [12], then $F = \hat{\Sigma}_{smp}$. Note that, if data has been spherized [13] the bandwidth structure F will correspond to the identity matrix, I, and F could simply be removed.

C. Model Compression

If samples are simply added without doing anything else, model complexity would increase linearly with the number of samples. This would be the same as saying that we save all previously observed samples, which defeats one of our initial goals.

One way to solve this problem is to compress the sample model when a certain threshold is met. The objective of the compression to approximate the original N-component sample distribution

$$p_s(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi_{\Sigma_i}(\mathbf{x} - \mathbf{x}_i) \tag{11}$$

by a M-component, $M < N$, equivalent $\hat{p}_s(\mathbf{x})$

$$\hat{p}_s(\mathbf{x}) = \sum_{j=1}^{M} \hat{w}_j \phi_{\hat{\Sigma}_j}(\mathbf{x} - \hat{\mathbf{x}}_j) \tag{12}$$

such that the resulting compressed KDE does not change significantly. This means that we are representing approximately same information using less components while and minimizing the induced error.

One way to compress the sample model is to resort to a clustering-based approach. The aim is to identify clusters of components in $p_s(\mathbf{x})$ such that each cluster can be sufficiently well approximated by a single component in $\hat{p}_s(\mathbf{x})$. Let $\Xi(M) = \{\pi_j\}_{j=1:\text{M}}$ be a collection of disjoint sets of indexes, which cluster $p_s(\mathbf{x})$ into M-sub-mixtures. The sub-mixture corresponding to indexes $i \in \pi_j$ is defined as

$$p_s(\mathbf{x}; \pi_j) = \sum_{i \in \pi_j} w_i \phi_{\Sigma_i}(\mathbf{x} - \mathbf{x}_i) \tag{13}$$

The parameters of the j-th component $\hat{w}_j \phi_{\hat{\Sigma}_j}(\mathbf{x} - \hat{\mu}_j)$ are defined by matching the first two moments (mean and covariance) of the sub-mixture π_j

$$\begin{aligned} \hat{w}_j &= \sum_{i \in \pi_j} w_i, \\ \hat{\mu}_j &= \hat{w}_j^{-1} \sum_{i \in \pi_j} w_i \mu_i, \\ \hat{\Sigma}_j &= \left(\hat{w}_j^{-1} \sum_{i \in \pi_j} w_i (\Sigma_i + \mu_i \mu_i^\intercal) \right) - \hat{\mu}_j \hat{\mu}_j^\intercal \end{aligned} \tag{14}$$

In fig. 1 an example is provided where the components of a sample distribution $p_s(\mathbf{x})$ are clustered to form another (compressed) sample distribution $\hat{p}_s(\mathbf{x})$. The KDEs of the original and compressed KDE are also depicted, showing that while the number of components in the sample distribution is reduced, the resulting KDE does not change significantly.

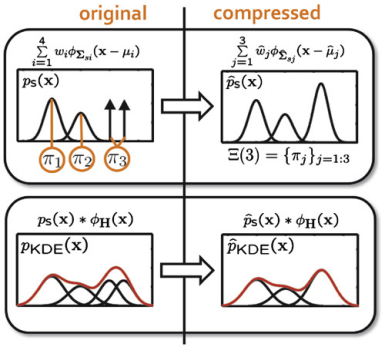

Fig. 1. Example of the compression of a four component sample distribution into a three component one. The upper row shows the sample distribution, and the lower one the corresponding KDE. Image taken from [9].

More formally, the compression seeks to identify the clustering assignment $\Xi(M)$ with the minimal number of clusters M, such that the error induced by each cluster is sufficiently low, i.e. , it does not exceed a predefined threshold D_{th}

$$\hat{M} = \arg\min_{M} E(\Xi(M)) \qquad \text{s.t. } E(\Xi(\hat{M})) \leq D_{th} \tag{15}$$

where $E(\Xi(M))$ is defined as the largest local clustering error $\hat{E}(p_s(\mathbf{x}; \pi_j), \mathbf{H}_{\text{opt}})$ under the clustering assignment $\Xi(M)$

$$E(\Xi(M)) = \max_{\pi_j \in \Xi(M)} \hat{E}(p_s(\mathbf{x}; \pi_j), \mathbf{H}_{\text{opt}}) \tag{16}$$

The local clustering error $\hat{E}(p_s(\mathbf{x};\pi_j),\mathbf{H}_{\text{opt}})$ corresponds to the error induced under the KDE with bandwidth $\mathbf{H}_{\text{opt}}$ if the cluster $p_s(\mathbf{x};\pi_j)$ is approximated by a single Gaussian. This error is defined next.

D. Local Clustering Error

Let us define $p_1(\mathbf{x})$ as a cluster, i.e., a sub-mixture of the sample distribution, as shown in eq(13). Also, consider $p_0(\mathbf{x})$ to be the resulting single Gaussian approximation described in 14. And finally, let $\mathbf{H}_{\text{opt}}$ to be the current estimation of the optimal bandwidth. We can define the local clustering error as the distance

$$\hat{E}(p_1(\mathbf{x}),\mathbf{H}_{\text{opt}}) = D(p_{1\text{KDE}}(\mathbf{x}),p_{0\text{KDE}}(\mathbf{x})) \tag{17}$$

between the corresponding KDEs

$$\begin{aligned} p_{1\text{KDE}}(\mathbf{x}) &= p_1(\mathbf{x}) * \phi_{\mathbf{H}_{\text{opt}}}(\mathbf{x}) \\ p_{0\text{KDE}}(\mathbf{x}) &= p_0(\mathbf{x}) * \phi_{\mathbf{H}_{\text{opt}}}(\mathbf{x}) \end{aligned} \tag{18}$$

One way to quantify the distance between distributions is using the Hellinger distance [14], which is defined as

$$\begin{aligned} &D^2(p_{1\text{KDE}}(\mathbf{x}),p_{0\text{KDE}}(\mathbf{x})) \triangleq \\ &\quad \tfrac{1}{2}\int \left(p_{1\text{KDE}}(\mathbf{x})^{1/2} - p_{0\text{KDE}}(\mathbf{x})^{1/2}\right)^2 d\mathbf{x} \end{aligned} \tag{19}$$

E. Distance between Mixture Models

While the Hellinger distance is a proper metric between distributions and is bounded to the interval $[0,1]$, it cannot be calculated analytically for mixture models. We calculate its approximation using the *unscented transform* [15], a special case of the Gaussian quadrature, which, similarly to Monte Carlo integration, relies on evaluating integrals using carefully placed points, called *sigma points*, over the support of the integral. As in Monte Carlo integration [16], an *importance* distribution is defined,

$$p_0(\mathbf{x}) = \gamma(p_1(\mathbf{x}) + p_2(\mathbf{x})) \tag{20}$$

which contains the support of both $p_1(\mathbf{x})$ and $p_2(\mathbf{x})$, with γ set such that $\int p_0(\mathbf{x})d\mathbf{x}$=1. We can then rewrite the Hellinger distance (19) into

$$D^2(p_1,p_2) = \frac{1}{2}\int g(\mathbf{x})p_0(\mathbf{x})d\mathbf{x} \tag{21}$$

where

$$g(\mathbf{x}) = \frac{\left(\sqrt{p_1(\mathbf{x})} - \sqrt{p_2(\mathbf{x})}\right)^2}{p_0(\mathbf{x})} \tag{22}$$

Note that the integral in 21 is an expectation over a nonlinearly transformed Gaussian random variable $\mathbf{x}$, and therefore admit to the unscented transform. According to [15] we can approximate $D^2(p_1,p_2)$ with

$$D^2(p_1,p_2) \approx \frac{1}{2}\sum_{i=1}^{N} w_i \sum_{j=0}^{2d+1} g({}^{(j)}\mathcal{X}_i)\,{}^{(j)}\mathcal{W}_i \tag{23}$$

where $\{{}^{(j)}\mathcal{X}_i, {}^{(j)}\mathcal{W}_i\}_{j=0:2d+1}$ are weighted sets of sigma points corresponding to the i-th Gaussian $\phi_{\Sigma_i}(\mathbf{x}-\mathbf{x}_i)$, and are defined as

$$\begin{aligned} {}^{(0)}\mathcal{X}_i &= \mathbf{x}_i, \qquad {}^{(0)}\mathcal{W}_i = \frac{k}{d+k} \\ {}^{(j)}\mathcal{X}_i &= \mathbf{x}_i + s_j\sqrt{(d+k)\xi_j}, \qquad {}^{(j)}\mathcal{W}_i = \frac{1}{2(d+k)} \\ s_j &= \begin{cases} 1, & j \le d \\ -1, & \text{otherwise} \end{cases} \end{aligned} \tag{24}$$

where $k = \max([0, m-d])$ and ξ_j is the j-th column of the matrix square root of Σ_i such that $\xi = \sqrt{\Sigma_i}$. Note that j was defined in the range $[0; 2d+1]$, and the square matrix ξ contains only d columns. This apparent notation confusion is cleared if we think that the sigma points are simply the mean $\mathbf{x}_i$ ($j=0$) and $\mathbf{x}_i \pm \xi_j$, i.e. , the sum and subtraction of each of the dimensions of the covariance matrix Σ_i. Therefore, the j-th column of ξ_j must be *counted* separately for the s_j=1 and s_j= $-$ 1 sets, such that $j = [1,d]$ for each set of sigma points. Moreover, let $\mathbf{UDU}^T$ be a singular value decomposition of covariance matrix Σ, such that $\mathbf{U} = \{U_1,\ldots,U_d\}$ and $\mathbf{D} = \text{diag}\{\lambda_1,\ldots,\lambda_d\}$, then $\xi_k = \sqrt{\lambda_k}U_k$.

For multidimensional systems, choosing $k = 3-d$ minimizes the mean squared error up to the fourth order. There is also no restriction on the sign of k, but if k is negative, then the distribution of the sigma points cannot be interpreted as a probability distribution. Furthermore, with a negative k there is the possibility that the resulting covariance will be non-positive semi-definite. In line with these discussions presented in [15] we set the parameter m to $m=3$ and force a non-negative value k resulting in $k = \max([0, 3-d])$.

F. Hierarchical Compression

The global optimization of (15) would require the evaluation all possible cluster assignments $\Xi(M)$, which, as the number of components grows, quickly becomes computationally prohibitive. A significant reduction in complexity can be achieved by a *hierachical* approach to clustering. Similar approaches have been successfully applied for data compression of Gaussian mixture models to a predefined number of components [4], [17]. In our implementation, we start by splitting the the entire sample distribution $p_s(\mathbf{x})$ into two sub-mixtures using Goldberger's K-means algorithm [4] for mixture models, with K=2. In order to avoid singularities associated with dirac-delta components of the sample model, instead of $p_s(\mathbf{x})$ we apply the K-means algorithm to the corresponding KDE model $p_{\text{KDE}}(\mathbf{x})$. Next, each sub-mixture is approximated by a single Gaussian using eq. 14 and the sub-mixture which yields the largest local error $\hat{E}(p_1(\mathbf{x}),\mathbf{H}_{\text{opt}})$ is further split into two sub-mixtures. This process is continued until the the largest local error is sufficiently small to fulfill the condition $E(\Xi(\hat{M})) \le D_{th}$ of eq. 15. This process produces a binary tree with $\hat{M}$ leafs and where each leaf represents the clustering assignments $\Xi(\hat{M}) = \{\pi_i\}_{j=1:M}$ among the components of the sample distribution $p_s(\mathbf{x})$. We can then use the cluster assignments $\Xi(\hat{M})$ to approximate the corresponding components in the sample distribution $p_s(\mathbf{x})$ by a single Gaussian, resulting in the desired compressed sample distribution $\hat{p}_s(\mathbf{x})$. Figure 2, illustrates this process.

Recall that we keep track of a detailed model for each component in the sample distribution. This means that if

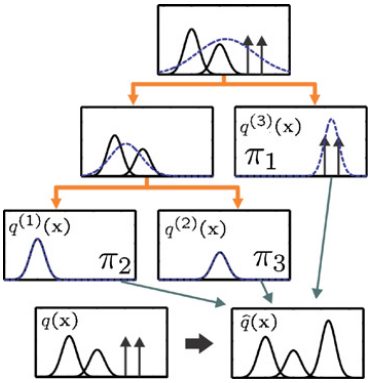

Fig. 2. Hierarchical Clustering example. Image taken from [10].

two components are merged, their underlying detailed models should be merged as well. The detailed model $\hat{q}_j(\mathbf{x})$ of the j-th component in the compressed model $\hat{p}_s(\mathbf{x})$ is calculated by first defining a normalized *extended* mixture model.

$$\hat{q}_{j\text{ext}}(\mathbf{x}) = \Big(\sum_{i\in\pi_j} w_i\Big)^{-1} \sum_{i\in\pi_j} w_i\, q_i(\mathbf{x}) \tag{25}$$

If the extended mixture has one or two components, then $\hat{q}_j(\mathbf{x}) = \hat{q}_{j\text{ext}}(\mathbf{x})$. If this mixture has more than two components, then the two-component detailed model $\hat{q}_j(\mathbf{x})$ is generated by splitting $\hat{q}_{j\text{ext}}(\mathbf{x})$ into two sub-mixtures using Golberger K-means and approximating each sub-mixture by a single Gaussian using eq. 14. The motivation for the two-component representation of the detailed model is that it provides the simplest possible model that enables detection and recovery of over-compressions. Detailed models with higher number of components would allow more detailed sample models and would better model the data. However this would be done at the cost of higher computational complexity.

G. Revitalization

The compression algorithm identifies and compresses clusters of components whose compression does not introduce a significant error into the KDE with bandwidth $\mathbf{H}_{\text{opt}}$ estimated at the time of compression. However, as more samples arrive, the sample distribution changes. Consequently, both $\mathbf{H}_{\text{opt}}$ and KDE estimation change. As a result, a compression which may have been valid for a KDE at some point in time, may later become invalid.

These over-compressions can be detected through the inspection of the *detailed model* of each component in the sample distribution $p_s(\mathbf{x})$. The local clustering error $\hat{E}(q_i(\mathbf{x}), \mathbf{H}_{\text{opt}})$ eq. 16 of each component in the sample distribution can be evaluated against its detailed model $q_i(\mathbf{x})$ to verify whether the error exceeds the threshold D_{th}. Those components in $p_s(\mathbf{x})$ for which $\hat{E}(q_i(\mathbf{x}), \mathbf{H}_{\text{opt}}) > D_{\text{th}}$, are removed from the sample distribution and replaced by the two components of their detailed model. These new components, however, need their own detailed model. For this, we generate a new detailed model based on the covariances of each of the new components. For example, let $w_i\phi_{\Sigma_i}(\mathbf{x}-\boldsymbol{\mu}_i)$ be one of the new components. If the determinant of Σ_i is zero, then this component is a Dirac-delta function, corresponding to a single data-point, and its detailed model is just the component itself. However, if the determinant is non-zero, it means that the component has been generated through clustering of several detailed models in previous compression steps. Its detailed model is initialized by splitting $\phi_{\Sigma_i}(\mathbf{x}-\boldsymbol{\mu}_i)$ along its principal axis [18] into a two-component mixture, whose first two moments match those of the original component. Splitting along the principal axis has two interesting properties: first, since the component is symmetric around the mean, the splitting process minimizes the error induced by splitting the Gaussian; and second, it is moment preserving, i.e., the mean and covariance of the split Gaussian, and thus, of the entire mixture, remains unchanged.

In detail, let $\mathbf{UDU}^T = \boldsymbol{\Sigma}_i$ be a singular value decomposition of $\boldsymbol{\Sigma}_i$ with singular values and singular vectors ordered by descending singular values. Then, the component detailed mixture model is defined as

$$\begin{aligned} q_i(\mathbf{x}) &= \sum_{k=1}^{2} w_k \phi_{\Sigma_k}(\mathbf{x}-\boldsymbol{\mu}_k), \\ \mu_1 = \mathbf{FM} + \mu_i, \qquad & \mu_2 = \mathbf{FM} - \mu_i, \\ \boldsymbol{\Sigma}_k = \mathbf{FCF}^{\mathrm{T}}, \qquad & w_k = \frac{1}{2} w_i \end{aligned} \tag{26}$$

where $\mathbf{C} = \text{diag}([3/4, \mathbf{0}_{1\times(d-1)}])$, $\mathbf{M} = [0.5, \mathbf{0}_{1\times(d-1)}]^{\mathrm{T}}$, $\mathbf{F} = \mathbf{U}\sqrt{\mathbf{D}}$ and $\mathbf{0}_{1\times(d-1)}$ represents an all-zeros row vector of length $(d-1)$.

H. Algorithm Overview

The previous sections formally described all individual phases of the online kernel density estimation approach, and a full overview of the method is now possible. As each new observation is added to the model, it is added in the form of a Dirac-delta function, i.e., a Gaussian with mean $\mathbf{x}$ and variance Σ=0. As more samples arrive, model complexity grows linearly. To avoid this and keep complexity low, a compression algorithm is called when complexity surpasses a predefined threshold. This threshold may be fixed, or allowed to change automatically during execution. The compression algorithm consists of two steps. The first revitalizes the mixture by splitting components that are no longer good representations of the underlying data. This allows us to have the highest fidelity possible in the model, prior to the second step. The second step merges components that are sufficiently similar, i.e., that represent approximately the same information and are thus redundant when considered together. These merges may later be invalidated by subsequent samples, and may be subject to the revitalization step. Figure 3 presents a graphical overview of this process.

III. XOKDE++ : Fast and Robust KDE

The online kernel density estimation approach presented before presents several desirable properties as it is non-parametric, does not require to save all previous observations, allows for model flexibility and evolution as more samples are added, and the theoretic algorithm is valid for arbitrary dimensionality.

However, the original oKDE implementation in MATLAB suffers from various problems as it is non-optimized MATLAB research code and highly redundant. This makes the task of

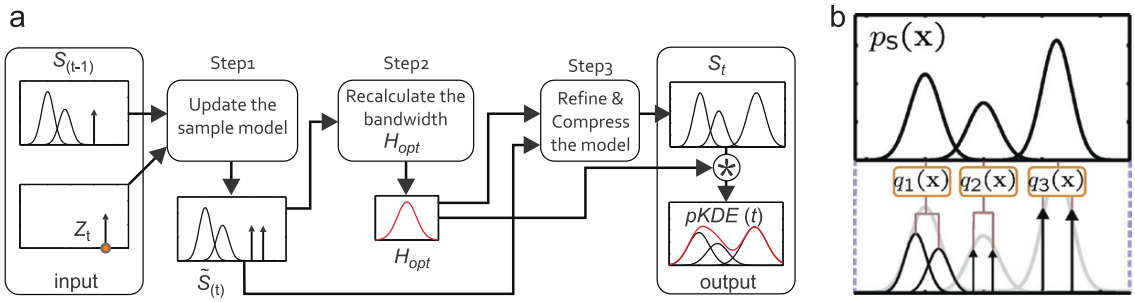

Fig. 3. Global overview of a full iteration of the oKDE approach (a). The sample model S_{t-1} is updated by a new observation $\mathbf{z}_t$ and compressed into a new sample model $S_(t)$. In (b), the new sample distribution along with its detailed model is shown. Image taken from [9].

changing and extending the algorithm much harder and error prone. Furthermore, as the number of feature dimensions grow, performance heavily degrades and memory usage grows prohibitively. Besides performance, the original implementation is also not designed for high dimensionality, which can lead to numeric instability. One such example is the covariance determinant, which is computed directly. For certain datasets it is easy to have situations where the determinant underflows or overflows.

The work presented in this paper presents an efficient and numerically robust, object oriented implementation of the oKDE. It was implemented in C++ and uses Eigen v3.2.2 [19] for algebraic computations such as, among others, matrix multiplication, singular value decomposition (SVD) and Eigenvalue Decomposition. Implementing this work in an object oriented programing (OOP) language allows for higher flexibility and abstraction advantages. Plus, the C++ allows for efficient memory usage by using move semantics, which effectively reduces the number of copies and memory allocations.

While we follow the theoretical model presented before, we took several implementation decisions in order to deal with numeric issues and coping with high dimensionality. Strategies to improve performance such as postponing computations until they are needed, or caching results until they are invalidated, are also described in the next subsections.

A. Numeric Stability

The gaussian probablity density function as defined in eq. 2 uses the determinant and the inverse of the covariance matrix. One way to efficiently compute the inverse of a non-singular matrix is to first compute it's LU decomposition and use it to compute the inverse. This is also convenient as we will need to compute the matrix determinant. With the LU decomposition, and since our covariance matrix is guaranteed to be positive definite, the determinant can be simply computed by multiplying the diagonal entries of the matrix U. This approach, however, may lead to numeric issues if we have a large number of dimensions. When working with 1000 or 5000 dimensions it is very easy to overflow or underflow even if we have double precision floating points. To avoid this pitfall, we compute the logarithm of the determinant instead. Again, the LU decomposition is convenient as this operation simply involves computing the logarithm of each entry in the diagonal of matrix U, and then sum those entries.

A covariance matrix is, by definition, a positive semi-definite matrix. This means that there is the possibility that sometimes the covariance matrix is singular. In that case the inverse cannot be computed. If the Gaussian is a Dirac-delta, this is expected. However, sometimes the data observations may have dimensions that change very seldom and we have not seen sufficient observations to have a different value in that dimension. This situation produces a rank deficient matrix, in which one dimension never changes, with a zero-valued covariance along that axis. To correct this situation we detect and correct singular or near singular matrices by computing it's eigen-decomposition and checking for eigenvalues bellow a predefined threshold of 10^-9. If any of the eigenvalues falls bellow, then a covariance revitalization algorithm is employed. This consists in correcting all problematic eigenvalues by 1% of the average of the eigenvalues. In other words, we inflate all flat dimensions to 1% the average variance in the other non-flat dimensions.

In both the optimal bandwidth and compression phases described in sections II-B and II-C are performed on a spherized, or whitened version of the data. This scales and decorrelates the data, reducing impact of sample distribution and outlier presence. This whitening transformation starts by approximating the whole model by a single Gaussian. The resulting covariance is the model covariance $\hat{\Sigma}$, we then compute the transformation parameters needed to transform the model covariance into the identity matrix. Those parameters are then individually applied to all components in the mixture. The inverse operation, the colouring transformation can be used to recover the original data. The whitening transform impacts the model in two ways. The first is performance, since in eq. (7) the F will be the identity matrix, and thus significantly reducing computational cost. The second advantage lies in the Goldberger K-Means approach in the Model Compression phase. Having the data normalized reduces the impact of outliers. It also removes dimensions with zero variance, leaving us with only a subspace. This helps the Goldberger K-Means converge to better solutions.

Since the whitening transformation is neutral, meaning we can transform forwards and backwards, we could apply the transformation to the data, and later apply the inverse operation. However, successive forward and backward operations might lead to small precision errors, that, with long computational times, could severely impact the estimated models. With this in mind a secondary whitened model is temporarily

created and used for the entire compression algorithm. In the revitalization phase we select the components that deviate too much from its internal underlying model. Those selected components are split in both main and secondary models. Then in the Hierarchical Clustering phase, the components will be grouped and marked for merge. This operation is performed on the secondary model. When complete, the marked components are actually merged on the main model, and the secondary model is discarded as it is no longer needed.

Sometimes, the observed data may produce optimal bandwidths of zero. This is inconvenient as it means the Diracs will remain without a covariance matrix leaving us unable to use the likelihood function. One solution for this, is to detect these occurrences and set the bandwidth as the identity matrix and then computing the backwards whitening transformation. Recall that the optimal bandwidth is computed on the whitened data.

B. Lazy Operation and Result Buffering

In theory, each time a sample is added, the optimal bandwidth should be updated. However, the optimal bandwidth is only needed for the Compression phase or to evaluate the likelihood of a given sample. If none of these operations is needed, then the optimal bandwidth computation can be postponed. What this means is that the cost of adding a sample reduces to simply adding one dirac component to the mixture and accordingly update the weights of the existing components. This can be done without any impact on model quality. An illustrative example can be that of adding a several of components and then performing the compression phase. Even if we did compute the optimal bandwidth on each component insertion, each subsequent one would replace the previous computed optimal bandwidth such that only the last would be used.

The determinant and the inverse of the covariance matrix of a component are needed to compute the likelihood function. This function is called very often, so computing the determinant and the inverse each time the likelihood function is needed is expensive and wasteful. A better way is to save the determinant and the inverse once it has been computed until the component covariance changes, making the previously computed inverse invalid. As with the optimal bandwidth, there is advantage in postponing the determinant and inverse computation until they are actually needed.

C. Diagonal Covariance Matrices

Using diagonal covariance matrices significantly reduces the amount of variables needed to be estimated. However, this forces the components to ignore dependencies across dimensions. In other words, we produce covariance matrices that ignore part of the data variance, producing less detailed components. This could pose a problem of accurately fitting the data, but can be compensated by using more components.

D. Extensibility

Besides speed and memory efficiency, one central contribution of our work is the adaptability and extensibility of the algorithm and implementation. Since it was implemented in a OOP language, in this case, C++, the templated classes allow for easy extension. For example, it is straightforward to replace full covariance by diagonal covariance. We can also extend the implementation to use triagular covariance matrices, by simply writing a class and call the template accordingly. The same logic applies if we want to replace the Gaussian pdf by a different one. Instead of Gaussians, we could easily use Student-t distributions simply by implementing the corresponding class and respecting the current interfaces. Similarly, changing the current clustering approach by another is also trivial as long as the interfaces are respected.

IV. Evaluation Setup

The goal of our work is to have an efficient implementation of the original oKDE [9] approach. Thus, we want to produce models with similar quality while taking a fraction of the time and memory needed by the original implementation. Even though the paper description is closely followed, there are implementation details that may be different from those of the original authors and thus, the final models are different. Plus, the algebra routines implemented in Eigen 3.x are not the same as the ones implemented in MATLAB. One example lies with the SVD computation, which produces and equally valid, but different decomposition. All of this has direct impact on the final model produced by the training algorithm and prevents us from producing exactly equal models.

However, having different models does not say anything about it's intrinsic quality. To properly evaluate our implementation and extensions, and to maximize the comparability of these results, we decided to follow the main strategies of the original oKDE paper [9]. Those are based on intrinsic and extrinsic evaluation. On the intrinsic side, we measure the quality by using the average negative log-likelihood and overall model complexity. On the extrinsic side, we measure accuracy on a classifier based task.

Besides the oKDE and xokde++, which are both generative methods, we decided to compare with a discriminative approach, which are typically better suited for classification tasks. Again with reproducibility of the original oKDE work in mind, we used the online independent SVM (OISVM) [20] as implemented in the DOGMA [21] MATLAB toolbox.

For all evaluations a variety of datasets was used, which correspond to the datasets used on the original paper. Some datasets have been updated in recent years, so, in order to have comparable results between our implementation and the original one, we ran both approaches in the same, most recent, datasets. In addition we also evaluate performance on our L^2f Face Database, which allows to test the high dimensional scenario, and for which neither the original oKDE implementation nor OISVM could handle.

As we want to, as much as possible, reproduce the results obtained in the original work of oKDE, we follow the same evaluation setup. Thus, for the online classifier training task, we randomly shuffled the data in each dataset and use 75% of the data to train and the remaining to test. For each dataset we generate 12 such random shuffles to minimize impact of lucky/unlucky partitions. To maximize comparability, the same 12 shuffles given to the matlab oKDE implementation are given to our C++ implementation.

For the kernel density estimation approaches each class is represented by the KDE model resultant of it's training samples. For the oKDE approach, each class model is initialized with 3 samples before adding the remaining 1 at a time. In our implementation there is no need for initialization and each model is trained by adding 1 sample at a time from the very start. In the end of the training phase we end up with k models, corresponding to k classes. To evaluate the classification performance we use a simple Bayesian criterion

$$\hat{y} = \arg\max_k p(\mathbf{x}|c_k)p(c_k) \tag{27}$$

where c_k corresponds to the kde trained with the samples from the k'th class, $p(\mathbf{x}|c_k)$ is the likelihood of the sample to be explained the k'th model, and $p(c_k)$ is the probability of a sample from c_k to occur in the entire $\hat{y}$ model.

OISVM is an online SVM approach which trains a binary classifier. Thus, for a multiclass problems we need to train k binary classifiers using all training samples corresponding to the class as positive examples, and the remaining training samples as negative examples. Then, for the classification assignment, a 1vsAll approach was followed

$$\hat{y} = \arg\max_k f_k(\mathbf{x}) \tag{28}$$

in which $f_k(\mathbf{x})$ provides the confidence score of the k'th class binary classifier for the sample $\mathbf{x}$. We used a second order (quadratic) polynomial kernel with gamma and coef0 parameters set to 1. The complexity parameter C was also set in advance and set to 1. A more proper way to set the C parameter is to use cross validation on the training set and find the most suitable parameter. However, our setup is that of online learning, were unknown classes and data is bound to appear. In a sense, not even a training set exists a priori, and the classifier is to be trained continuously as more samples arrive. Preliminary experiments also shown that for several datasets, changing the C parameter did not change the result significantly, and it typically worsened results. For these reasons, and since the risk of harming classifier performance significantly appears to be contained, we set C=1 parameter for all experiments.

As stated before, we wish to make minimal assumptions about data nature and distribution, as both can change as more samples arrive. As such, we purposely do not make any preprocessing such as feature normalization. The data is used as is available in the dataset, and any potential data transformation is the responsibility of the internals of each approach.

A. UCI Datasets

The authors of the original oKDE paper used several freely available datasets to evaluate and compare their approach against other state of the art approaches. In the spirit of having comparable results, we will use the same datasets where possible, as some datasets are no longer available or have been updated. These real-life datasets are part of the UCI Machine Learning Repository [22] and differ in length, data dimensionality and number of classes. Table IV-A presents a summarized view of the dataset's properties and Fig. 4 shows dataset balancing.

Dataset	N_S	N_D	N_C
Iris	150	4	3
Yeast	1484	8	10
Pima	768	8	2
Winequality-red	1599	11	6
Winequality-white	4898	11	7
Wine	178	13	3
Letter	20000	16	26
Image Segmentation (seg)	2310	19	7
Steel	1941	27	7
Breast Cancer	569	30	2
Skin	245057	3	2
Covtype	581012	10	7

TABLE I. PROPERTIES OF DATASETS WITH REAL-LIFE DATA, DENOTED BY NUMBER OF SAMPLES (N_S), NUMBER OF DIMENSIONS (N_D) AND NUMBER OF CLASSES (N_C)

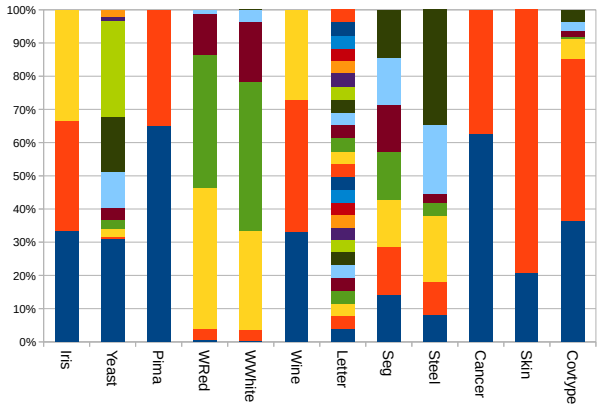

Fig. 4. Class balancing accross all UCI datasets.

B. L^2f Face Database

The L^2f Face Database consists of 30000 indoor free pose face images from 10 subjects, corresponding to 3000 images per subject. The capture was performed using a PlayStation Eye camera with a 640x480px resolution at 60Hz. Each video frame was fed to the open-source computer vision tool OpenCV [23] face detector, which uses a Haar Cascade [24] approach for face detection. This detector predominantly detects frontal face poses, but is tolerant to high inclination and head rotation poses. The detected face square regions were cropped and resized to 64x64px blocks with pixel mixing using the pamscale tool. In the capture process, each subject was asked to stand in front of the camera and make a variety of facial poses, expressions, to speak with other person and to move the head in several directions. In essence, each speaker was asked to behave normally, while avoiding being static. Figure 5 shows some examples on the variety of poses in the dataset. By letting the OpenCV face detection algorithm define the region where the face is located, to later crop the frame, we obtain a fully automatic face capture phase, and since we have no restriction on facial pose or expression, no related post-processing is necessary. This capture approach, being simple, automatic and robust (on facial pose and expression), means it can be easily integrated in an online training algorithm such as ours, where the images even need not to be saved in disk, being directly fed to the training algorithm.

However, to have reproducible results, a fixed dataset had to be created and its images stored. With that in mind, this

capture phase resulted in a free pose dataset captured in indoor conditions and in a realistic face capture scenario, allowing us to test the online density estimation approach in a face recognition task. The images from the dataset, being automatically cropped from the the OpenCV Haar Cascade detected face region, present some background. One way to remove the background is to take advantage of the fact that faces tend to be centered in the cropped square, such that removing a number of pixels from the four sides results in a restricted view of the face with minimal or no background. To evaluate the impact of background presence, five crops are available (0,2,4,8 and 16). A crop of 0 means no extra crop was performed, *i.e.* , the original image. A crop of 2, means that 2 pixels were discarded from each side of the squared image. Subsequent crops of 4,8 and 16 follow the same logic. Figure 6 presents an example of the impact of the various crops in various sample images.

Fig. 5. Some examples of the dataset pose and expression variability.

Fig. 6. All crop schemes, 0, 2, 4, 8, 16, and the masking approach to extract 128 informative pixels.

V. Results

In the following subsections the performance of the original oKDE, xokde++ and OISVM will be evaluated under various perspectives. Furthermore, one advantage of our implementation is it's modularity. Thus, it is very easy to implement a version that uses diagonal covariance matrices instead of full covariance. This, although potentially worsening the model quality, as relationships across dimensions are ignored, has dramatic impact on speed and memory usage, especially as the number of dimensions increases. While for full covariance the number of variables to estimate grows quadratically with the number of dimensions, in the diagonal version that number grows linearly. In order to study how much the model quality degrades by using diagonal covariance matrices, we also present results for the *okde++/diag*.

A. Model Quality

To measure the estimation quality, we use the average negative log-likelihood. We also compare of the average complexity of the models. Table II presents results for experiments after all training samples were observed. The xokde++ implementation produces models with similar complexity but with lower average negative log-likelihood. This is indicative that, depending on the dataset data, we produce better fits with approximately the same complexity. This is more evident on datasets, such as the steel, segmentation and cancer, with very dissimilar feature spaces which translates in dimensions with very little variance, and others with very large variances. Our numerical stability contribution allows to recover from these situations by correcting the covariances with a small induced variance, computed using available information, until more data arrives and allows for a more accurate estimation.

Dataset	xokde++	xokde++/diag	oKDE
	Complexity		
Iris	28 ± 3	22 ± 3	28 ± 3
Yeast	31 ± 15	30 ± 19	31 ± 15
Pima	62 ± 9	42 ± 19	64 ± 7
Winequality-red	39 ± 23	53 ± 37	40 ± 24
Winequality-white	31 ± 24	54 ± 40	36 ± 25
Wine	44 ± 7	44 ± 7	45 ± 7
Letter	65 ± 12	42 ± 9	66 ± 14
Image Segmentation (seg)	49 ± 12	51 ± 24	52 ± 11
Steel	8 ± 2	20 ± 7	4 ± 1
Breast Cancer	40 ± 7	153 ± 11	50 ± 12
Skin	8 ± 2	10 ± 2	8 ± 2
Covtype	18 ± 5	17 ± 5	20 ± 5
	$-\mathcal{L}$		
Iris	7.4 ± 1.7	3.3 ± 1.3	7.8 ± 1.6
Yeast	-13.2 ± 0.5	-14.3 ± 0.3	-7.1 ± 1.7
Pima	29.3 ± 0.5	27.6 ± 0.2	31.0 ± 0.7
Winequality-red	-0.2 ± 0.6	0.2 ± 0.3	14.7 ± 4.2
Winequality-white	0.5 ± 0.2	1.6 ± 0.2	55.2 ± 54.3
Wine	51.1 ± 4.3	33.0 ± 3.4	64.9 ± 18.5
Letter	17.9 ± 0.1	19.9 ± 0.1	18.2 ± 0.1
Image Segmentation (seg)	25.7 ± 0.8	31.0 ± 1.0	156.6 ± 33.7
Steel	–	85.2 ± 4.3	–
Breast Cancer	-25.6 ± 5.7	-16.4 ± 2.4	433.9 ± 33.7
Skin	13.5 ± 0.3	13.9 ± 0.2	13.4 ± 0.2
Covtype	51.9 ± 0.1	55.9 ± 0.0	52.6 ± 0.1

TABLE II. The average number of components per model and average negative log-likelihood $-\mathcal{L}$ along with ± one standard deviation. The empty entries in the steel dataset mean that the model did not converge properly.

B. Classifier Accuracy

To analyse the discriminative properties of the implementations we constructed classifiers as described in section IV. As show in table V-B, we achieve better performance in 7 out of 12 datasets, and lower performance in only 3 datasets. Some of the results for oKDE are different than those reported in the original papers. These differences are accounted by the fact that no dataset was balanced or normalized for unit variance, as reported in the original paper. We intentionally used the datasets with no further processing such as it provides the most realistic online operation scenario, where data sample distribution is now known. These results provide further indication that our implementation possesses greater numeric stability, handling non-normalized datasets.

Results for the online independent support vector machine (OISVM) are also provided as it is usefull to compare the discriminative power of our generative approach with a discriminative one. As the table shows, OISVM typically outperforms the generative approaches, but not by far. It also

	xokde++	xokde++/diag	oKDE	OISVM
Iris	96.4 ± 2.7	95.0 ± 3.4	96.4 ± 2.4	97.1 ± 0.2
Yeast	49.7 ± 2.3	48.1 ± 1.6	50.6 ± 3.3	59.2 ± 7.7
Pima	67.8 ± 3.4	70.1 ± 3.9	69.7 ± 2.9	76.9 ± 1.5
Winequal-red	62.0 ± 2.5	54.6 ± 1.9	56.9 ± 6.3	58.3 ± 10.6
Winequal-white	49.9 ± 1.3	42.4 ± 1.7	44.9 ± 10.6	53.2 ± 37.2
Wine	97.7 ± 1.4	98.5 ± 1.8	93.9 ± 6.1	96.8 ± 0.3
Letter	95.8 ± 0.2	93.4 ± 0.4	95.8 ± 0.2	93.0 ± 0.4
Image Seg.	91.5 ± 1.1	89.4 ± 1.2	75.0 ± 5.3	95.0 ± 84.1
Steel	31.9 ± 11.0	56.9 ± 9.0	8.7 ± 1.0	24.3 ± 0.0
Breast Cancer	94.8 ± 1.7	96.2 ± 1.7	52.8 ± 12.0	95.9 ± 0.6
Skin	99.6 ± 0.1	99.4 ± 0.0	99.7 ± 0.1	99.8 ± 0.0
Covtype	52.0 ± 1.2	51.6 ± 0.6	68.0 ± 0.9	—

TABLE III. RESULTS FOR AVERAGE CLASSIFICATION ACCURACY ALONG WITH ± ONE STANDARD DEVIATION.

shows that it suffers from the same numeric instabilities as the oKDE implementation.

C. Time and Memory Performance

The previous results shown that our implementation produces models with similar or better quality than the original one. However, our implementation does so at lower memory cost and much faster computation time. Table IV presents results for time and memory usage taken to train and test the datasets. It shows that our approach achieves speedups of 3 to 10 times depending on the dataset. For the diagonal matrices the speedup is even grater, going from 11 to up to 40 times. Note that the oKDE MATLAB implementation already has calls to vectorized and parallel routines. Still, our serial version, which uses a single core for the whole computation time, is able to consistently achieve speedups. In terms of memory usage, when compared to the oKDE we use, at most, 10% of the memory required by the original implementation. This difference is more critical in large datasets such as the skin and covtype. For these datasets, the oKDE implementation needed 913MB and 5064MB, respectively, as our needed only 83MB and 361MB.

For the OISVM, the maximum memory usage was 2255MB, which is even more than the needed for oKDE. In fact, OISVM, has a tendency to need more memory. This trend is even clearer as the number of classes increases. Since the OISVM trains a binary classifier, for multiclass problems we need to train K classifiers. The letter dataset is an example for that, since it has 26 classes the memory needed relative to the oKDE was 4 times more. For large datasets time performance also degrades, where the skin dataset took nearly twice the time it took for oKDE. This is even more evident the covtype dataset, which has 500,000 samples, and 20 days of computation were not enough to complete training a single shuffle, let alone the 12 shuffles from our evaluation setup.

D. Performance of Full vs Diagonal Covariances

The previous results show that our implementation produces consistently equivalent or better models than the oKDE original implementation at lower computational and memory cost. However, our implementation also allows to choose between full and diagonal covariances. Table VI presents a detailed comparison of the relative difference of the impact of using diagonal covariances in the quality of the model and computational requirements. From this, we can see that using diagonal covariances produces models with slightly higher complexity, with small loss in model quality. In fact, for the steel dataset, using diagonal covariances allowed the model to avoid the severe numeric instability found when using full covariances, where feature variance systematically produced singular matrices.

The speedup relative to the full covariance approach ranges between 3 to 22 times with a median value of 5.5. For full covariances, the number of dimensions affect the memory requirements and the number of variables increases quadratically with the number of dimensions. So datasets with higher dimensionality will have greater benefit for using diagonal covariances. However, the diagonal approach tends to need more components, this, sometimes, partially offsets the previous gain. Even so, the median memory usage of the diagonal approach is 78% of the needed for full covariance. This number is even greater if we only consider datasets with more than 15 dimensions. In this case the median value falls to 58% with only small losses in accuracy, around 2% absolute if we ignore the 25% in the steel dataset. Considering all datasets, except the steel dataset, which is indicative for the diagonal approach numeric robustness but not usefull for an aggregate estimate of quality loss of using diagonal covariances, we obtain a median value of 1.35%. This means that the discriminative model quality for the covariance matrices is very similar to the full ones.

VI. CONCLUSIONS

In this paper we presented a state of the art online kernel density estimation approach and implementation that is numerically robust and able to handle high dimensionality. It achieves comparable model quality when compared with the original oKDE implementation while needing significantly less memory and computational time. The numerical stability improvements provide xoke++ the ability to cope with non-normalized data and achieve a median classifier accuracy of 79.6%, which compares with 68.9% of the original oKDE implementation. This represents a 15.5% median improvement over the baseline approach. Compared to the implementation used in the state of the art oKDE papers, xoke++ is up to 40 times faster, needs 90% less memory, has greater numerical robustness, and produces, on average, models with higher quality. Furthermore, our implementation, by being written in a object oriented language presents desirable properties such as extensibility and flexibility. This allow for extra research on extending the algorithm to be performed with less effort. One example is the diagonal covariance matrices, which were easily implemented and provide aditional efficiency in time and memory usage while sacrificing little model quality. Although our models discriminative power is slightly inferior to the discriminative approach of OISVM, our implementation is more stable and computationaly efficient.

With this implementation, it is now possible to apply the online kernel density estimation approach in high dimensional data. This has direct implications to fields such as sensor management [25] and big data, where it now becomes possible to continuously capture and model high dimensional data samples using the KDE approach. The xokde++ implementation has also shown dramatic computational performance improvements over the baseline oKDE. Even so, there is still room for improvement as some of the components of the algorithm, such

as bandwidth computation and model compression, are serious bottlenecks. This issue is particularly evident when working with high dimensions.

Future lines of work include both computational performance and numerical stability improvements. Regarding computational performance, in many applications using dense matrices is inefficient, especially in high dimensional tasks where the feature space is very sparse. One way to avoid this is to use approaches that make use of the sparsity of the feature space [26], [27], [28]. On the numerical stability two main lines can be followed. First, we could avoid correcting the degenerate covariances, which always introduces some ϵ error. For this we could use the degenerate Gaussian probability density function and computing the pseudo-inverse and pseudo-log-determinant [29]. Secondly we could to experiment with Toeplitz covariance matrices [30], [31]. However, using sparse or Toeplitz matrices may cause problems in the estimation of the log-determinant. Cai et. al. [32] approach the problem of estimating of the log-determinant in sparse matrices and prove that consistent estimation of the log-determinant is not possible when $p > n$, where p is the number of dimensions and n the sample size.

REFERENCES

[1] D. Reynolds, "Gaussian mixture models," *Encyclopedia of Biometrics*, pp. 659–663, 2009.

[2] V. Melnykov and I. Melnykov, "Initializing the em algorithm in gaussian mixture models with an unknown number of components," *Computational Statistics & Data Analysis*, vol. 56, no. 6, pp. 1381–1395, 2012.

[3] M. Song and H. Wang, "Highly efficient incremental estimation of Gaussian mixture models for online data stream clustering," in *Defense and Security*. International Society for Optics and Photonics, 2005, pp. 174–183.

[4] J. Goldberger and S. T. Roweis, "Hierarchical clustering of a mixture model," in *Advances in Neural Information Processing Systems*, 2004, pp. 505–512.

[5] S. Chen, X. Hong, and C. J. Harris, "Probability density estimation with tunable kernels using orthogonal forward regression," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 40, no. 4, pp. 1101–1114, 2010.

[6] U. Ozertem, D. Erdogmus, and R. Jenssen, "Mean shift spectral clustering," *Pattern Recognition*, vol. 41, no. 6, pp. 1924–1938, 2008.

[7] B. Han, D. Comaniciu, Y. Zhu, and L. S. Davis, "Sequential kernel density approximation and its application to real-time visual tracking," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, no. 7, pp. 1186–1197, 2008.

[8] A. Declercq and J. H. Piater, "Online learning of Gaussian mixture models - a two-level approach." in *VISAPP (1)*, 2008, pp. 605–611.

[9] M. Kristan, A. Leonardis, and D. Skočaj, "Multivariate online kernel density estimation with Gaussian kernels," *Pattern Recognition*, vol. 44, no. 10, pp. 2630–2642, 2011.

[10] M. Kristan and A. Leonardis, "Online discriminative kernel density estimator with gaussian kernels," *Cybernetics, IEEE Transactions on*, vol. 44, no. 3, pp. 355–365, 2014.

[11] M. P. Wand and M. C. Jones, *Kernel smoothing*, ser. Monographs on statistics and applied probability. Chapman & Hall/CRC, 1994, vol. 60.

[12] T. Duong and M. Hazelton, "Plug-in bandwidth matrices for bivariate kernel density estimation," *Journal of Nonparametric Statistics*, vol. 15, no. 1, pp. 17–30, 2003.

[13] B. W. Silverman, *Density estimation for statistics and data analysis*. CRC press, 1986, vol. 26.

[14] D. Pollard, *A user's guide to measure theoretic probability*. Cambridge University Press, 2002, vol. 8.

[15] S. J. Julier and J. K. Uhlmann, "A general method for approximating nonlinear transformations of probability distributions," RRG, Department of Engineering Science, University of Oxford, Tech. Rep., 1996.

[16] E. Veach and L. J. Guibas, "Optimally combining sampling techniques for monte carlo rendering," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM, 1995, pp. 419–428.

[17] A. T. Ihler, "Inference in sensor networks: Graphical models and particle methods," Ph.D. dissertation, Massachusetts Institute of Technology, 2005.

[18] M. Huber, *Nonlinear Gaussian Filtering: Theory, Algorithms, and Applications*. KIT Scientific Publishing, 2015, vol. 19, pp. 108–109.

[19] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[20] F. Orabona, C. Castellini, B. Caputo, L. Jie, and G. Sandini, "On-line independent support vector machines," *Pattern Recognition*, vol. 43, no. 4, pp. 1402–1412, 2010.

[21] F. Orabona, *DOGMA: a MATLAB toolbox for Online Learning*, 2009, software available at http://dogma.sourceforge.net.

[22] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[23] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[24] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001, pp. I–511.

[25] M. Huber, *Probabilistic Framework for Sensor Management.* KIT Scientific Publishing, 2009, vol. 7.

[26] H. Liu, J. D. Lafferty, and L. A. Wasserman, "Sparse nonparametric density estimation in high dimensions using the rodeo," in *International Conference on Artificial Intelligence and Statistics*, 2007, pp. 283–290.

[27] A. Krishnamurthy, "High-dimensional clustering with sparse gaussian mixture models," 2011.

[28] M. Azizyan, A. Singh, and L. Wasserman, "Efficient sparse clustering of high-dimensional non-spherical gaussian mixtures," in *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*. (AISTATS) San Diego, CA, USA. JMLR: W&CP volume 38, 2015, to appear.

[29] P. Mikheev, "Multidimensional gaussian probability density and its applications in the degenerate case," *Radiophysics and quantum electronics*, vol. 49, no. 7, pp. 564–571, 2006.

[30] J. Pasupathy and R. Damodar, "The gaussian toeplitz matrix," *Linear algebra and its applications*, vol. 171, pp. 133–147, 1992.

[31] T. T. Cai, Z. Ren, and H. H. Zhou, "Estimating structured high-dimensional covariance and precision matrices: Optimal rates and adaptive estimation," *The Annals of Statistics*, vol. 38, pp. 2118–2144, 2014.

[32] T. T. Cai, T. Liang, and H. H. Zhou, "Law of log determinant of sample covariance matrix and optimal estimation of differential entropy for high-dimensional gaussian distributions," *Journal of Multivariate Analysis*, vol. 137, pp. 161–172, 2015.

Dataset	xokde++	ρ	xokde++/diag	ρ	oKDE	OISVM	ρ
Iris	0.5 ± 0.1	11.0	0.1 ± 0.02	34.2	5.0 ± 0.7	0.2 ± 0.1	26.7
Yeast	39.4 ± 3.2	4.5	6.5 ± 0.7	27.1	177.2 ± 8.1	7.7 ± 0.3	23.1
Pima	16.9 ± 2.1	7.9	3.3 ± 0.3	40.1	133.9 ± 2.3	1.5 ± 0.4	90.1
Winequality-red	61.0 ± 7.7	5.4	14.4 ± 0.7	22.6	326.4 ± 27.5	10.6 ± 1.8	30.7
Winequality-white	130.7 ± 4.0	5.8	39.0 ± 1.7	19.6	764.1 ± 51.3	37.2 ± 3.6	20.5
Wine	3.0 ± 0.5	3.2	0.5 ± 0.1	21.4	9.7 ± 0.6	0.3 ± 0.1	29.4
Letter	2119.0 ± 66.7	3.0	191.7 ± 3.5	33.7	6452.9 ± 191.9	5225.4 ± 1341.0	1.2
Image Segmentation (seg)	296.8 ± 26.1	1.7	45.0 ± 6.1	11.5	515.7 ± 21.5	84.1 ± 12.0	6.1
Steel	191.4 ± 36.5		8.7 ± 0.5		44.0 ± 2.9	0.02 ± 0.05	1834.1
Breast Cancer	52.6 ± 6.9	2.6	11.7 ± 1.1	11.6	135.6 ± 11.2	0.6 ± 0.1	221.8
Skin	572.9 ± 204.5	11.5	192.2 ± 40.9	34.2	6565.1 ± 1471.0	10522.5 ± 2144.0	0.6
Covtype	9803.8 ± 1614.9	2.6	1156.6 ± 124.5	22.2	25713.3 ± 2081.3	—	—

TABLE IV. RESULTS FOR TIME PERFORMANCE ALONG WITH ± ONE STANDARD DEVIATION. TIME IS PRESENTED IN SECONDS. SPEEDUPS OVER THE BASELINE OKDE APPROACH ARE SHOWN IN COLUMN ρ .

Dataset	xokde++	ρ	xokde++/diag	ρ	oKDE	OISVM	ρ
Iris	4.6	3.76%	4.5	3.68%	123.0	116.6	94.84%
Yeast	6.8	3.93%	5.8	3.35%	173.1	148.9	86.04%
Pima	6.2	3.94%	5.2	3.29%	157.5	124.8	79.25%
Winequality-red	8.1	5.60%	6.2	4.24%	145.4	162.2	111.53%
Winequality-white	10.3	5.05%	8.3	4.08%	204.3	228.4	111.79%
Wine	6.3	4.38%	4.8	3.32%	144.3	120.2	83.32%
Letter	42.6	9.83%	23.2	5.36%	433.3	1823.9	420.94%
Image Segmentation (seg)	15.7	7.72%	7.7	3.75%	203.9	192.3	94.33%
Steel	10.5	5.77%	7.2	3.97%	181.3	194.5	107.29%
Breast Cancer	10.6	5.50%	6.6	3.40%	193.1	141.8	73.43%
Skin	83.1	9.10%	83.2	9.11%	913.3	2255.1	246.93%
Covtype	361.2	7.13%	360.4	7.12%	5064.8	—	—

TABLE V. RESULTS FOR MEMORY PERFORMANCE ALONG WITH RELATIVE USAGE COMPARED TO THE BASELINE OKDE APPROACH, SHOWN IN COLUMN ρ. MEMORY IS PRESENTED IN MEGABYTES.

Dataset	xokde++					xokde++/diag								
	acc	K	-$\mathcal{L}$	time	mem	acc	K	-$\mathcal{L}$	time	memory	AAccD	K diff	TS	mem use
Iris	96.4%	28	7.4	0.5	4.6	95.0%	22	3.3	0.1	4.5	-1.4%	-7	3.1	97.9%
Yeast	49.7%	31	-13.2	39.4	6.8	48.1%	30	-14.3	6.5	5.8	-1.6%	-1	6.0	85.2%
Pima	67.8%	62	29.3	16.9	6.2	70.1%	42	27.6	3.3	5.2	2.3 %	-21	5.1	83.5%
Winequal-red	62.0%	39	-0.2	61.0	8.1	54.6%	53	0.2	14.4	6.2	-7.4%	14	4.2	75.8%
Winequal-white	49.9%	31	0.5	130.7	10.3	42.4%	54	1.6	39.0	8.3	-7.5%	22	3.4	80.8%
Wine	97.7%	44	51.1	3.0	6.3	98.5%	44	33.0	0.5	4.8	0.8 %	0	6.6	75.8%
Letter	95.8%	65	17.9	2119.0	42.6	93.4%	42	19.9	191.7	23.2	-2.4%	-23	11.1	54.5%
Image Seg.	91.5%	49	25.7	296.8	15.7	89.4%	51	31.0	45.0	7.7	-2.2%	2	6.6	48.7%
Steel	31.9%	8	–	191.4	10.5	56.9%	20	85.2	8.7	7.2	24.9%	12	22.0	68.8%
Breast Cancer	94.8%	40	-25.6	52.6	10.6	96.2%	153	-16.4	11.7	6.6	1.5 %	113	4.5	61.8%
Skin	99.6%	8	13.5	572.9	83.1	99.4%	10	13.9	192.2	83.2	-0.1%	3	3.0	100.1%
Covtype	52.0%	18	51.9	9803.8	361.2	51.6%	17	55.9	1156.6	360.4	-0.4%	-2	8.5	99.8%

TABLE VI. COMPARISON OF THE EFFECT OF USING DIAGONAL COVARIANCES ON ACCURACY (ACC), NUMBER OF COMPONENTS IN THE MODEL (K), NEGATIVE LOG-LIKELIHOOD (-$\mathcal{L}$), TIME, AND MEMORY. THE LAST FOUR COLUMNS PRESENT THIS COMPARISON. A NEGATIVE VALUE ON THE ABSOLUTE ACCURACY DIFFERENCE (AACCD) MEANS A LOSS IN ACCURACY. THE TIME SPEEDUP (TS) COLUMN REPRESENTS THE SPEED UP, WHERE A VALUE OF 6 MEANS THE PROGRAM RAN 6 TIMES FASTER. THE MEMORY USE COLUMN DISPLAYS THE RELATIVE MEMORY USAGE OF USING DIAGONAL COVARIANCES VS FULL. A VALUE OF 80% MEANS THAT THE DIAGONAL APPROACH USED 80% OF THE MEMORY USED BY THE FULL COVARIANCE APPROACH.