Incremental Learning of Probabilistic Models in High-Dimensional Space

Jaime Rodrigues Ferreira

Thesis to obtain the Master of Science Degree in Information Systems and Computer Engineering

Supervisors:  Doctor David Manuel Martins de Matos  
              Doctor Ricardo Daniel Santos Faro Marques Ribeiro

Examination Committee

Chairperson:  Doctor Daniel Jorge Viegas Gonçalves
Supervisor:   Doctor David Manuel Martins de Matos
Member of the Committee:  Doctor Pedro Filipe Zeferino Tomás

October 2015
Acknowledgements

I would like to express my gratitude to my advisor, Prof. David Matos, for truly being an exemplary professor, who encourages and brings out the best in his students. I thank him for giving me the opportunity to continuously work with him since I was still an undergraduate student. If I am a better and more skilfully student, I owe it to him. Finally, I thank him for the support, discussions and patience he had during this thesis.

I want to thank my co-advisor Prof. Ricardo Ribeiro, for helping me with all those miscellaneous math related questions. Furthermore, I would like to thank Instituto Superior Técnico as a whole, and particularly several professors whom I encountered throughout my education, and that were excellent teachers. I also thank Pedro Moreira, for all his interest and patience when we talked about my thesis.

Finally but not least, I would like to thank my wife Inês, for all her support and patience, and for holding up “just one more month”, five times. The most special gratitude however, goes to my little princess daughter Carolina for her unwavering cheerfulness and for showing me the true meaning of dedication.

Lisboa, November 10, 2015
Jaime Rodrigues Ferreira
For my wife and daughter…
A capacidade de modelar dados é central em todas as tarefas de aprendizagem automática. Muitas abordagens operam em modo batch, e requerem que os dados estejam disponíveis à priori. Contudo, em alguns domínios, pode haver necessidade de construir e actualizar um modelo conforme os dados chegam. Em domínios de big data os dados podem nem sequer caber em memória, tornando abordagens batch inadequadas. Abordagens online são mais adequadas para estas tarefas devido a não necessitarem que os dados estejam disponíveis à priori. Algumas abordagens vão mais longe e não necessitam de guardar os dados anteriormente observados. Nesta tese descrevemos e implementamos uma abordagem estado da arte baseada em online kernel density estimation, que permite modelar dados continuamente construindo e mantendo um modelo de mistura de Gaussianas. A nossa implementação, que denominamos de xokde++, foi também desenhada para suportar alta dimensionalidade e distribuições de dados arbitrárias, tais como dados degenerados ou enviesados. Além disso, o código é extensível e fácil de integrar em projectos maiores. Comparando com o estado da arte actual, a nossa implementação é até 40 vezes mais rápida, necessita de 90% menos memória, tem maior estabilidade numérica e produz, em média, modelos com melhor qualidade. Para mostrar a extensibilidade e melhorar a estabilidade numérica implementamos também uma variante do xokde++ que substitui os kernels Gaussianos de matrizes de covariância completa por kernels Gaussianos de matrizes diagonais, obtendo resultados positivos tanto em estabilidade numérica como em performance computacional.
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.
Palavras Chave

Aprendizagem Automática
Estimação de Densidade
Alta Dimensionalidade
Aprendizagem Incremental
Estabilidade Numérica

Keywords

Machine Learning
Density Estimation
High Dimensionality
Incremental Learning
Numerical Robustness
## 4 xokde++: Fast and robust KDE

### 4.1 Numeric Stability

#### 4.1.1 Determinant Computation

#### 4.1.2 Degenerate Covariance Matrices

#### 4.1.3 Whitening Transformation

#### 4.1.4 Optimal Bandwidth Stability

### 4.2 Diagonal Covariance Matrices

### 4.3 Lazy Operation and Result Buffering

### 4.4 Code Extensibility and Integration

### 4.5 Summary

## 5 Evaluation Setup

### 5.1 UCI Datasets

### 5.2 High Dimensional Task

#### 5.2.1 Final considerations

### 5.3 L2F Face Database

### 5.4 Summary

## 6 Results and Discussion

### 6.1 Experimental Setup

### 6.2 Model Quality

### 6.3 Classifier Accuracy

### 6.4 Time and Memory Performance

### 6.5 Performance of Full vs Diagonal Covariances

### 6.6 Face Recognition Task

### 6.7 Summary

## 7 Conclusion

## A Covariance Properties

## B Whitening Transformation
List of Tables

2.1 Sample size required for accurate KDE estimation in different dimensionality . . . 11

5.1 Properties of the UCI datasets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45

6.1 Average negative log-likelihood and model complexity . . . . . . . . . . . . . . 56
6.2 Classification Accuracy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
6.3 Time Performance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
6.4 Memory Performance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
6.5 Full vs Diagonal covariance performance . . . . . . . . . . . . . . . . . . . . . . . 67
6.6 Classification Accuracy on a face recognition task . . . . . . . . . . . . . . . . . . 69
Introduction

The ability to model data is central to all machine learning tasks. In most tasks, the 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 more extreme scenarios, the data may change over time, and thus is desirable to build and maintain a model of current and past data. The model should be capable of evolving to account for the new information without getting stuck in some past representation, i.e., to allow some degree of forgetfulness. This is critical in long-running applications where we want continuously receive and model data.

Our goal is to be able to continuously model data, with minimal restrictions regarding data dimensionality and distribution. Moreover, we want this model to allow us to build semantic relationships. These can be as simple as being able to distinguish two entities, to something like measuring similarities between those entities. Ideally, this process would also be completely unsupervised, entities would be automatically discovered, and the model would evolve as more information arrives. However, this falls out of scope for our work, and this thesis is solely concerned with the task of accurately modeling those entities.

Continuously modeling data with arbitrary dimensions for long periods of time poses several challenges. 1) How to avoid retraining the whole model as more samples arrive? 2) How to avoid keeping all previously observed samples? 3) How to deal with arbitrary data dimension and structure? 4) How to recover from numeric pitfalls? As the model needs to keep training even if by some reason something went wrong. 5) How to use the data model to infer semantics?

To demonstrate and evaluate the approach used to handle these issues, we use several real life datasets that measure the modeling quality of approach. However, a more challenging setup, one which tests arbitrary dimension and data distribution is needed. For this, we collected a face dataset in uncontrolled conditions, such as facial pose, and expression. This provides a high dimensional dataset with a high number of samples collected in a realistic way. Although our task is not facial recognition, this is also a field where the problem of high dimensionality is often handled and may provide us with insights on how to tackle high dimensionality in our solution.

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, i.e., a probabilistic description of how the data is distributed. 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 common approach to train GMMs is to use the Expectation Maximization algorithm.
CHAPTER 1. INTRODUCTION

(EM) (Reynolds 2009). However, this approach suffers from two problems. First, the number of Gaussian components must be defined a priori, which limits applicability in long term modeling. Furthermore, it suffers from initialization issues, where an improper choice may lead to poor models, unable to capture the complete structure of the underlying probability density function (pdf). This is typically solved by using training data to find the best initialization (Melnykov and Melnykov 2012). 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, which might have been optimal for the original data, sub optimal for the current data. Finally, although incremental versions of EM exist (Song and Wang 2005), they still need to keep parts of the previously observed data and to define, a priori, the target number of components.

As defined before, our setup is one where the data to be modeled is not known a priori, and where the shape of the target distribution might change 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 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 an unfeasible approach. Furthermore, since the goal is to model data continuously and indefinitely, we want our model to be able to evolve as new data appears while making possibly obsolete, older data, 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 a single component in the mixture. This implies that the complexity grows linearly with the number of observed samples. A solution for this is to compress the model either to a predefined number of components (Goldberger and Roweis 2004), or to optimize some data-driven choice (Chen, Hong, and Harris 2010). Ozertem, Erdogmus, and Jenssen (2008) 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 in the estimated models to properly generalize to unobserved data and to adjust model complexity and parameters without directly accessing all previously observed samples. Han, Comaniciu, Zhu, and Davis (2008) 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 (2008) 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 handle the problem of non-Gaussian data by using a uniform distribution to represent regions where data is non-Gaussian.

Kristan, Leonardis, and Skočaj (2011) and more recently Kristan and Leonardis (2014), 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 (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 an online model, it can be updated in single sample up-
dates, 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 modelling multivariate data while making little assumptions on the nature of that data. More importantly, it contains a forgetting factor, through which old information progressively loses weight in the model. This allows continuously modeling and adapting to new information, accounting for 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, previous merges may be no longer valid. To retain the 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.

One of our goals is to be able to keep and maintain 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 solution is strongly related with the algorithm from Kristan, Leonardis, and Skočaj (2011) and recently from Kristan and Leonardis (2014). However, our requirements forces us to be concerned with computational performance and numeric stability. Plus, we aim to improve model quality, so a flexible and easily extensible solution was also a requirement.

Thus, our work 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.

In the following chapters, we present related work in Gaussian mixture estimation (Chapter 2), starting with the traditional parametric expectation-maximization approach, and then introducing non-parametric kernel density estimation. In addition we also visit a few other approaches to online estimation of Gaussian mixture models, including some mentioned in this introduction. In Chapter 3 we present Kristan, Leonardis, and Skočaj (2011) approach to online kernel density estimation, along with our concerns regarding computational speed and numeric stability. Chapter 4 describes implementation decisions to solve issues with computational speed and numerical stability, identified on the previous chapter. Chapter 5 describes the evaluation setup to evaluate the model quality. It describes the datasets, including a new face dataset, created to test the high dimensionality scenario in this thesis. It also presents a small survey of the features and approaches typically used in face recognition. The reason this literature is interesting is because face recognition systems typically operate in high dimensional scenarios and a wide variety of features are available. Surveying this literature may help find strategies to cope with high dimension. Chapter 6 presents and discusses the evaluation results under multiple perspectives. Chapter 7 presents conclusions and future lines of work, such as strategies to further improve computational performance and model quality.
In an unsupervised learning problem, the goal is to find hidden structure in unlabeled data. Since the provided examples are unlabeled, there is no error or reward signal to help guide the learner to reach an ideal solution. In multidimensional data, this is a very difficult problem, as it consists of finding natural groupings, i.e., clusters, that can have very different shapes and sizes, demanding a flexible and powerful modeling tool. There are multiple clustering techniques, depending on different constraints and data they are meant to receive. K-means (MacQueen et al. 1967) is a popular example of such a system that organizes data around centroids. Being both simple and easy to compute, it is able to retain an acceptable performance for a wide range of tasks, such as vector quantization (Jegou, Douze, and Schmid 2011) and feature learning. However, this approach gives us hard partitions between the centroids chosen to be the representative of a given class. In particular, it does not allow modeling uncertainty, because we are only measuring distances and not densities. Clustering Ensembles (Vega-Pons and Ruiz-Shulcloper 2011) make use of multiple clustering algorithms and try to reach a consensus, in order to improve the quality of individual data clusterings. The consensus can be as simple as a majority voting (Lam and Suen 1997). This type of approach requires multiple concurrent clustering algorithms that have to be fused after every one has converged, making it a potentially non-ideal solution for an online system.

2.1 Gaussian Mixture Models

Gaussian Mixture Models (GMMs) (Reynolds 2009) have shown to be a powerful tool in approximating various distributions which may be far from Gaussian (Wand and Jones 1994). They are able to model uncertainty as each individual Gaussian has a probability density function that represents the data distribution in a fuzzy way. Modeling uncertainty is important because, in an unsupervised scenario, the samples are not labeled, so we are never sure to which class they belong, we can only have a certain degree of certainty, i.e., there is a high probability that the observation belongs to a given cluster. A GMM is a parametric probability density function represented as a weighted sum of individual Gaussian component densities. Using GMMs allows us to make no assumptions over the observed data, while feeding them unlabeled data. The only limitation comes from feature choice, which should be discriminative of the observation, since the model’s descriptive power depends entirely on the actually observed data features.

Formally, a Gaussian mixture is a weighted sum of M component Gaussian probability density functions (pdf) as given by the equation,

\[
p(x|\theta) = \sum_{i=1}^{M} w_i g(x|\mu_i, \Sigma_i)
\]  

(2.1)
where \( x \) is a \( D \)-dimensional data vector (i.e., the observation features), \( \theta \) is the Gaussian mixture parameters, \( w_i, i = 1, \ldots, M \) are the mixture weights, and \( g(x|\mu_i, \Sigma_i), i = 1, \ldots, M \), are the component Gaussian pdfs. Each component is a \( D \)-variate Gaussian probability density function of the form,

\[
g(x|\mu_i, \Sigma_i) = \frac{1}{2\pi^{D/2} |\Sigma_i|^{1/2}} \exp \left\{ \frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) \right\}
\]  

(2.2)

with mean vector \( \mu_i \) and covariance matrix \( \Sigma_i \). The sum of the mixture weights is constrained such that \( \sum_{i=1}^{M} w_i = 1 \). The Gaussian mixture model is parametrized by the mean vectors, the covariance matrices and component weights as shown in eq. 2.3.

\[
\theta = w_i, \mu_i, \Sigma_i \quad i = 1, \ldots, M
\]  

(2.3)

We can also choose to use the full covariance matrix, or constrain it to be diagonal. This later case has the advantage of being less computationally demanding, but we are bound to lose information if the features are not statistically independent. However, since the component Gaussians are collaborating in modeling the overall feature density, full covariance matrices might not be necessary since the linear combination of the Gaussians diagonal covariance are capable of modeling the correlations between feature vectors. This approach, although leading to a potential loss of accuracy (mitigated by adding more components), is advantageous as we get a less computationally demanding problem. Magdon-Ismail and Purnell (2009) proposed an approximation of the full covariance by using low-rank perturbed matrices. This approach allows for a comparable efficiency to the diagonal approximation while retaining the ability to capture correlations between dimensions.

GMMs are often used in biometric systems, and are well matured in the fields of speaker verification and language recognition (Reynolds and Rose 1995; Campbell, Sturim, and Reynolds 2006; Kim, Kim, Lim, and Seo 2010). Their applicability in other fields such as robotics and computer vision has also been tried with good results (Gross, Yang, and Waibel 2000; Li, Hua, Lin, Brandt, and Yang 2013).

### 2.2 Batch Estimation

One way to estimate the parameters of the GMM, represented by \( \theta \) and defined in eq. 2.3, is using the maximum likelihood (ML) estimation. The aim of ML is to find the model parameters that maximize the likelihood of the GMM, given the training data. For a sequence of observations \( T \) with features \( X = \{x_1, \ldots, x_T\} \), the GMM likelihood, assuming independence between observations\(^1\), can be written as shown in eq. 2.4.

\[
p(X|\theta) = \prod_{t=1}^{T} p(x_t|\theta)
\]  

(2.4)

This expression is a non-linear function of the parameters \( \theta \) and direct maximization is not possible. However, ML parameter estimates can be obtained using the expectation-maximization (EM) algorithm (Dempster, Laird, Rubin, et al. 1977). The basic idea of the EM algorithm is, beginning with an initial model \( \theta \), to estimate a new model \( \bar{\theta} \) such that \( p(X|\bar{\theta}) \geq p(X|\theta) \). This

\(^1\)The independence assumption is often incorrect but needed to make the problem tractable.
new model becomes the initial model for the next iteration and the process is repeated until some convergence threshold is reached.

Thus, in the EM algorithm we can start with some initial estimate $\theta$ (e.g. random) and perform the E-Step, which consists of computing the membership weights for all data points $T$ in all mixture components $M$. This a posteriori probability for component $i$ is defined by equation eq. 2.5.

$$Pr(i|x_t, \theta) = \frac{w_i g(x_t|\mu_i, \Sigma_i)}{\sum_{k=1}^{M} w_k g(x_t|\mu_k, \Sigma_k)} \quad 1 \leq i \leq M \quad 1 \leq t \leq T$$ (2.5)

Note that the weights $w_i$ for each point are a membership probability, thus constrained by $\sum_{k=1}^{M} w_{ik} = 1$. This yields an $T \times M$ matrix of membership weights, where each row sums to 1. Then, we can use these weights to calculate new parameter values in the M-Step, (eqs. 2.6, 2.7, and 2.8)

**Mixture Weights**

$$\bar{w}_i = \frac{1}{T} \sum_{t=1}^{T} Pr(i|x_t, \theta) \quad 1 \leq i \leq M$$ (2.6)

**Means**

$$\bar{\mu}_i = \frac{\sum_{t=1}^{T} Pr(i|x_t, \theta)x_t}{\sum_{t=1}^{T} Pr(i|x_t, \theta)} \quad 1 \leq i \leq M$$ (2.7)

**Variances**

$$\bar{\Sigma}_i = \frac{1}{T_i} \sum_{t=1}^{T} w_{ti} (x_t - \bar{\mu}_t)(x_t - \bar{\mu}_t)^\top \quad 1 \leq i \leq M$$ (2.8)

### 2.2.1 Initialization and Convergence issues for EM

The EM algorithm can be initialized with a set of parameters and then conducting the E-step, or by starting with a set of initial weights and then doing the M-step. These initial parameters or weights can be chosen randomly, or via some heuristic method, such as using K-means, to cluster the data first and define the means based on those memberships. After each iteration is complete, we can compute the log-likelihood to decide if we have converged to, at least, a local minimum. We do this by checking whether the likelihood has not improved in a significant manner from the previous iteration to the current one. Using the log-likelihood is useful because it allows us to define the overall log-likelihood as the sum of the individual components log-likelihoods, as defined by eq. 2.9.

$$\log l(\theta) = \sum_{t=1}^{T} \log p(x_t|\theta) = \sum_{t=1}^{T} \left( \log \sum_{i=1}^{M} w_i g(x_t|\mu_i, \Sigma_i) \right)$$ (2.9)
2.2.2 Outlier sensibility and issues with high dimensional space

One good property of GMMs is its relative robustness to outliers. This is justified because we are fitting the model to the observed data while maximizing the overall likelihood. With this, the weight of a few outliers, provided they are much fewer than the normal data, will not affect the overall model in a significant way. However, if data is very noisy, then the outliers will be much more frequent and may increase the chance of the EM algorithm converging to local minimum.

Unfortunately, one major drawback of the GMM approach is the computational complexity when dealing with high dimensional space. In a 1-D scenario, for each component, $\mu$ and $\Sigma$ are scalars. Generically, the model can handle a $D$ dimensional space. This means that, as described before, $\mu$ is a vector with $D$ entries, and $\Sigma$ is a matrix of $D \times D$ size. Having the complexity grow quadratically with the number of dimensions is a major limitation when using full covariance matrices. This can be mitigated by having a common covariance matrix for all components, or have them be diagonal. However, we will lose some information, since the correlations between dimensions will be lost, but an increased number of components can compensate for that loss (Reynolds 2009).

Training GMMs using EM suffers from the aforementioned initialization and convergence issues. These can be avoided by using an entirely different approach to train GMMs. The next section will describe this approach.

2.3 Kernel Density Estimation

Kernel density estimation is a non-parametric approach to estimate the probability density function of a random variable. Silverman (1986) provides a detailed introduction to kernel density estimation and its difficulties.

This approach does not suffer from the EM initialization problems as it does not require any prior the definition of number of components. The kernel density estimator is defined as

$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

(2.10)

where $h$ is a smoothing parameter, and $K$ is the kernel, defined as a non-negative function that integrates to 1

$$\int_{-\infty}^{\infty} K(x) \, dx = 1$$

(2.11)

According to Silverman (1986), $K$ will usually, but not always, be the normal density function, which is a symmetric probability density function. Other kernels are available such as the uniform, triangular, and Epanechnikov. The later is optimal in a mean square error sense (Epanechnikov 1969) although the loss of efficiency is small for the other mentioned kernels. Due to its convenient mathematical properties, the normal kernel is often used, with $K(x) = \phi(x)$, where $\phi$ is the standard normal density function. Figure 2.1 presents an example of a kernel density estimation, along with the individual kernels.

One central element of this method is the smoothing parameter, or bandwidth. Figure 2.2 depicts the effect of different values of the bandwidth on an univariate kernel density estimation.
2.3. KERNEL DENSITY ESTIMATION

Figure 2.1: Example of a kernel density estimation, along with individual kernels.

For a larger number of dimensions, a similar effect is expected, as large values of $h$ may lead to oversmoothed density estimations that fail to capture the underlying sample distribution.

2.3.1 Difficulties in high-dimensional space

Intuition gained from experience in two or three dimensions may be misleading when considering data in higher-dimensional spaces. One example, provided by Silverman (1986), is the importance of tails in high dimensions. Consider the effect of resetting $f(x)$ to zero at all points of $f(x) < 0.01$, and rescaling the resulting function to be a probability density, i.e., for its integral to sum to 1. Since very few data will fall in regions where $f$ is very small, the effect on its truncated version is relatively low. In ten dimensions, this is no longer true. Consider the short tailed and well-behaved standard normal distribution, with zero mean an unit variance. For this distribution, over half the observations will, on average, fall at points where the density $f(x)$ is less than 0.01 times its maximum value. The reason for this is that if $X$ is the standard ten-variate normal then

$$f(X)/f(0) = \exp(-\frac{1}{2}X^T X) \sim \exp(-\frac{1}{2} \chi^2_{10})$$

and since the median of $\exp(-\frac{1}{2} \chi^2_{10})$ is 9.34, then the median of $f(X)/f(0)$ is $\exp(-9.34/2)=0.0094$. The innocuous truncation applied for the univariate case, will have a large effect for the ten-variate case. Paradoxically, large regions of high density may be devoid of observations, for a moderately sized sample number. Scott and Thompson (1983) call this the empty space phenomenon. In the ten-dimensional example we can show that 99% of the mass is at points whose distance from the origin is greater than 1.6. This constrasts with the univariate case, where nearly 90% of the mass lies between ±1.6. From this, we can conclude
that it is likely to be difficult to accurately estimate the density, except from enormous sample sizes. Table 2.1 illustrates how quickly the required sample size increases with the number of dimensions. This table shows the required sample number to ensure a relative mean squared error lower than 0.1, when estimating a standard multivariate normal density using a normal kernel and a bandwidth that minimizes the mean squared error.

### 2.4 Online Estimation of GMMs

As mentioned in the introduction, we want a model capable of evolving and adapting to new observations as they occur. This includes changing the model parameters and possibly adding more or less components to maximize the likelihood of the model. Also, since we aim for an online system that will be getting information continuously, we want to avoid having to keep all past observations: otherwise, we would run out of memory or end up with performance issues after a while.

We define the three main challenges as: (i) Managing the model complexity, i.e., the number of components, that should increase or decrease according to the data observed so far; (ii) the model should be updated without explicitly having to access all previous observations; (iii) the computational effort needed for a single update should not depend on the amount of data seen so far and the model should be compact and not grow significantly with the number of observations.

Several attempts to develop online approaches have been tried, solving one or more of the aforementioned challenges (Arandjelovic and Cipolla 2006; Calinon and Billard 2007; En-
### 2.4. Online Estimation of GMMS

<table>
<thead>
<tr>
<th>Dimensionality</th>
<th>Required sample size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>67</td>
</tr>
<tr>
<td>4</td>
<td>223</td>
</tr>
<tr>
<td>5</td>
<td>768</td>
</tr>
<tr>
<td>6</td>
<td>2790</td>
</tr>
<tr>
<td>7</td>
<td>10700</td>
</tr>
<tr>
<td>8</td>
<td>43700</td>
</tr>
<tr>
<td>9</td>
<td>187000</td>
</tr>
<tr>
<td>10</td>
<td>842000</td>
</tr>
</tbody>
</table>

Table 2.1: Approximate sample size required to ensure that the relative mean square error of estimating a standard multivariate normal density is less than 0.1. Table reproduced from (Silverman 1986).

Song and Wang (2005) use EM with model selection to estimate the mixture of the new block of data and statistical tests to merge with existing components. One drawback is that they assume data comes in blocks.

Arandjelovic and Cipolla (2006) proposed an extension of EM with split and merge rules, which allows a single data point at a time, instead of blocks. However, they assume that the distances between consecutive data points are sufficiently small, which may not hold true for generic applications.

Engel and Heinen (2011) approach to manage model complexity proposes the use of a minimum likelihood criterion to decide if a new observation vector $x$ can be represented by any of the mixture component's or a new component has to be added to the mixture. For each incoming data point, the algorithm verifies whether it minimally fits any mixture component. A data point is not recognized as belonging to a component $k$ if its probability $p(x|k)$ is lower than a previously specified minimum likelihood threshold. If it is rejected by all components, meaning it bears new information, a new component is added to the model, adjusting its parameters. This minimum likelihood (or novelty) threshold affects the sensibility of the learning process to new concepts, with higher thresholds generating more concepts. Having to fine tune this parameter can be difficult, as it will depend on the task and the data provided. Given the major impact on the overall performance of the algorithm, the authors suggest specifying the minimum value for the acceptable likelihood, $\tau_{nov}$, as a fraction of the maximum value of the likelihood function. Hence, a new mixture component is created when the data point $x = (x_1, \ldots, x_D)$ matches the novelty criterion presented in eq. 2.13.

$$p(x|i) < \frac{\tau_{nov}}{(2\pi)^{D/2} \sqrt{|\Sigma_i|}} \quad 1 \leq i \leq M$$ (2.13)

where $M$ is the number of components. The authors also propose a compact representation of past observations, thus avoiding having to keep all data points. They argue that one can use the sum of posterior probabilities of the data presented so far, belonging to component $i$, as
a measure of confidence about the estimates. When a new data point $x$ arrives, the posterior probability is computed and variable $s_{pi}$ is updated, as described in eq. 2.14.

$$s_{pi} = s_{pi} + p(i|x)$$ \hspace{1cm} (2.14)

Variable $s_{pi}$ is then used to update $\mu_i$, $\Sigma_i$, and $w_i$ (eqs. 2.15, 2.16, and 2.17)

$$\mu_i = \mu_i + \frac{p(i|x)}{s_{pi}}(x - \mu_i)$$ \hspace{1cm} (2.15)

$$\Sigma_i = \Sigma_i - (\mu_i - \mu_i^{old})(\mu_i - \mu_i^{old})^\top + \frac{p(i|x)}{s_{pi}}[(x - \mu_i)(x - \mu_i)^\top - \Sigma_i]$$ \hspace{1cm} (2.16)

$$p(i) = \frac{s_{pi}}{\sum_{k=1}^{M}s_{pk}}$$ \hspace{1cm} (2.17)

In eq. 2.16, $\mu_i^{old}$ refers to $\mu_i$ at time $t - 1$, i.e., before updating.

Kristan, Skoˇcaj, and Leonardis (2010) naively add components to the mixture and, when a threshold is surpassed, a compression algorithm is applied. They propose that the problem of modeling samples by a probability density function can be viewed as a problem of kernel density estimation. If a set of $n_t$ samples $\{x_i\}_{i=1}^{n_t}$ were to be observed at once at some time $t$, then the kernel density estimate would have kernels placed at locations $x_i$ with equal standard deviations, or bandwidths $h_t$

$$\hat{p}_t(x; h_t) = \frac{1}{n_t} \sum_{i=1}^{n_t} K_{h_t}(x - x_i)$$ \hspace{1cm} (2.18)

where $K_{h_t}(x - x_i)$ is a Gaussian kernel $g(x|x_i, h_t)$. We want $\hat{p}_t(x; h_t)$ to be as close as possible to the underlying distribution $p(x)$ that generated the samples. One way to measure that closeness is by using the mean integrated squared error (MISE) (eq. 2.19).

$$\text{MISE} = E[\hat{p}_t(x; h_t) - p(x)]^2$$ \hspace{1cm} (2.19)

Then, applying a Taylor expansion, assuming a large sample set and Gaussian kernels (which is true in our scenario), they write the asymptotic MISE (AMISE) (eq. 2.20).

$$\text{AMISE} = \frac{1}{2\sqrt{\pi}}(h_mt)^{-1} + \frac{1}{4}h_t^4R(p''(x))$$ \hspace{1cm} (2.20)

In eq. 2.20, $p''(x)$ is the second derivative of $p(x)$ and $R(p''(x)) = \int p''(x)^2dx$. Minimizing AMISE with respect to bandwidth $h_t$ gives the AMISE-optimal bandwidth.

$$h_t\text{AMISE} = \left[2\sqrt{\pi}R(p''(x))n_t\right]^{-1/3}$$ \hspace{1cm} (2.21)

Thus, for a new observation we can start from a known pdf $\hat{p}_{t-1}(x)$, from the previous time step, and calculate a new pdf $\hat{p}_t(x)$ by adding the newly created kernel,

$$\hat{p}_t(x) = (1 - \frac{1}{n_t})\hat{p}_{t-1}(x) + \frac{1}{n_t}K_{h_t}(x - x_t)$$ \hspace{1cm} (2.22)

In order to obtain the optimal $h_t$, we need to solve eq. 2.21. The problem is that we cannot obtain it exactly, since it depends on $p(x)$, which is the unknown distribution we are trying to approximate. However, the current estimate of $p(x)$ is initialized using the distribution from
2.4. ONLINE ESTIMATION OF GMMS

the previous time step \( \hat{p}(x) \approx \hat{p}_{t-1}(x) \). With this, an approximation of the optimal bandwidth can be obtained (eq. 2.23) by approximating \( p(x) \approx \hat{p}(x) \) and applying eq. 2.21.

\[
\hat{h}_t = \left[ 2\sqrt{\pi} R(\hat{p}_t(x)) n_t \right]^{-1/5}
\]  
(2.23)

The resulting kernel \( K_{\hat{h}_t}(x - x_t) \) is then combined with \( \hat{p}_{t-1}(x) \) into an improved estimate of the unknown distribution,

\[
\hat{p}_t(x) = (1 - \frac{1}{n_t})\hat{p}_{t-1}(x) + \frac{1}{n_t}K_{\hat{h}_t}(x - x_t)
\]  
(2.24)

Next, the improved estimate \( \hat{p}_t(x) \) from eq. 2.24 is plugged back into equation (2.23) to re-approximate \( h_t \). These two equations are iterated until convergence, which usually happens in five iterations (Kristan, Skočaj, and Leonardis 2010). One problem with this approach is that each observation adds one component to the mixture. In order to maintain low complexity, the authors propose a compression algorithm initiated whenever the number of components exceeds a value \( N_{\text{comp}} \). This compression algorithm does not force removing components, it merely tries to remove some. In this sense, \( N_{\text{comp}} \) only determines the frequency at which the compression is called. This value can be fixed at a predefined value: a large one if we want to prevent unnecessary calls to compression; or it can be allowed to vary. The authors used a simple rule to dynamically adjust \( N_{\text{comp}} \): If no components are removed in the compression step, \( N_{\text{comp}} \) is increased, \( N_{\text{comp}} \leftarrow c_{\text{scale}}^{-1} N_{\text{comp}} \). Otherwise, if the number of remaining components falls below \( c_{\text{scale}}^{-1} N_{\text{comp}} \), the threshold is decreased, \( N_{\text{comp}} \leftarrow c_{\text{scale}}^{-1} N_{\text{comp}} \). For all experiments, a \( c_{\text{scale}} \) value of 1.5 was used. A summarized description of the algorithm is described in Algorithm 1.

**Input:** \( \hat{p}_{t-1}(x), x_t \ldots \) the initial density approximation and the new sample  
**Output:** \( \hat{p}_{t-1}(x), x_t \ldots \) the new approximation of density

1: Initialize the current distribution \( \hat{p}(x) \approx \hat{p}_{t-1}(x) \).
2: Estimate the bandwidth \( h_t \) of \( K_{\hat{h}_t}(x - x_t) \) according to (2.23) using \( \hat{p}_t(x) \).
3: Re-estimate \( \hat{p}_t(x) \) according to (2.24) using \( K_{\hat{h}_t}(x - x_t) \).
4: Iterate steps 2 and 3 until convergence.
5: If the number of components in \( \hat{p}_t(x) \) exceeds a threshold \( N_{\text{comp}} \), compress \( \hat{p}_t(x) \) using the compression algorithm (described below).
6: If required, adjust the threshold \( N_{\text{comp}} \).

**Algorithm 1:** Online kernel density estimation

In the compression algorithm, we want to approximate the reference mixture \( p(x) \) with a simplified approximation \( \hat{p}(x|\theta) \). Since the reference mixture is known, the difference between \( p(x) \) and \( \hat{p}(x|\theta) \) can be measured by the integrated squared error (ISE)

\[
ISE(\theta) = \int (p(x) - \hat{p}(x|\theta))^2 dx.
\]  
(2.25)

The problem of finding an equivalent to \( p(x) \) can thus be posed as seeking an optimal \( \theta \) while minimizing the ISE (eq. 2.26).

\[
\hat{\theta} = \arg \min \left[ \int \hat{p}^2(x|\theta) dx - 2E_{p(x)}\{\hat{p}(x|\theta)\} \right],
\]  
(2.26)
where \( E_{p(x)} \{ \hat{p}(x|\theta) \} \) is the expectation with respect to the reference distribution \( p(x) \), and \( \theta = \{ w_i, \mu_i, \sigma_i \}_{i=1}^N \) with \( N \) being the number of components.

Minimizing the ISE, with \( E_{p(x)} \{ \hat{p}(x|\theta) \} \) kept fixed, equals to minimizing the first term \( \int \hat{p}^2(x|\theta)dx \). Expanding this term yields a weighted sum of expectations of pairs of components of \( \hat{p}(x|\theta) \)

\[
\int \hat{p}^2(x|\theta)dx = \sum_{i=1}^N \sum_{j=1}^N w_i w_j c_{ij},
\] (2.27)

where \( c_{ij} = \int K_{\sigma_i}(x - \mu_i)K_{\sigma_j}(x - \mu_j)dx \). The expectations among non-overlapping components will yield low values of \( c_{ij} \), while expectations among overlapping ones will yield high values of \( c_{ij} \). In this case, we can minimize ISE by assigning small weights to the overlapping components and large weights to non-overlapping ones. We can say that the term in eq. 2.27 is sparsity-inducing, as it prefers selecting those components that are far apart.

We can see that optimizing the ISE between \( p(x) \) and \( \hat{p}(x|\theta) \) will result in a subset of components in \( \hat{p}(x|\theta) \) by selecting components in high-probability regions of \( p(x) \), while preferring configurations in which the selected components are far apart. The authors rewrote the minimization of the ISE into a classical quadratic program and solved it with a variant of a Sequential Minimal Optimization (SMO) scheme (Schölkopf, Platt, Shawe-Taylor, Smola, and Williamson 2001). The compression algorithm has two main steps: reduction and organization. In the reduction step, a subset of components from \( \hat{p}(x|\theta) \) is removed using SMO. In the next step, organization, the error between \( \hat{p}(x|\theta) \) and the reference \( p(x) \) is reduced by optimizing eq. 2.26 using a Levenberg-Marquardt (LM) optimization (Marquardt 1963). These two steps are iterated, gradually reducing the complexity of \( \hat{p}(x|\theta) \), until convergence, i.e., when no more components are removed.

The number of components removed at each iteration can be controlled by inflating the variances of \( \hat{p}(x|\theta) \) by some inflation parameter \( \alpha > 1 \) before applying the SMO. For a large \( \alpha \), many components will overlap significantly and thus be removed. However, removing too many components increases the error in the approximation of \( p(x) \). Thus, \( \alpha \) must be defined such that the inflated pdf \( \hat{p}(x|\theta) \) is always close enough to the reference \( p(x) \). They achieve this by adjusting \( \alpha \) such that the Hellinger distance (Pollard 2002; Kristan, Skočaj, and Leonardis 2010) between \( p(x) \) and \( \hat{p}(x|\theta) \), \( H(p(x), \hat{p}(x|\theta)) \), is always smaller than some predefined distance \( H_{\text{dist}} \).

Note that, in the organization step, it is not necessary to optimize \( \hat{p}(x|\theta) \) until convergence, one only needs to reduce the distance between \( \hat{p}(x|\theta) \) and \( p(x) \) to be smaller than the \( H_{\text{dist}} \). This will allow for a set of components in \( \hat{p}(x|\theta) \) to overlap after the inflation and, thus, be removed in the reduction step. The authors used five LM iterations in each organization step. Full parameter (\( \theta \)) optimization is only needed after the reduction-organization iterations converge. The compression algorithm is then finished.

Declercq and Piater (Declercq and Piater 2008) have a different approach. 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. This is achieved by having a two-level mixture model. In
2.4. ONLINE ESTIMATION OF GMMS

At the first level, we have a precise GMM from the observations up to time $t$:

$$p^t(x) = \frac{\sum_{i=1}^{N} \pi_i^t g(x|\mu_i^t, C_i^t)}{\sum_{i=1}^{N} \pi_i^t},$$

(2.28)

where each Gaussian has its weight $\pi_i^t$, mean $\mu_i^t$, and covariance $C_i^t$. When a new data point arrives, it is represented by its distribution $g^t(x|\mu^t, C^t)$. The new GMM is computed in two steps:

1. **Concatenate** - produce a model with $N + 1$ components by combining the GMM and the new data point.

2. **Simplify** - if possible, merge some of the Gaussians to reduce the complexity of the GMM.

The first step is simply:

$$p^t(x) = \frac{\sum_{i=1}^{N} \pi_i^{t-1} g(x|\mu_i^{t-1}, C_i^{t-1}) + \pi^t g^t(x|\mu^t, C^t)}{\sum_{i=1}^{N} \pi_i^{t-1} + \pi^t}$$

(2.29)

In the second step, the goal is to reduce model complexity while still giving a precise description of the observations. To estimate the fidelity $\lambda$ of a Gaussian model, one must first compute the distance between the model and the corresponding data set. The authors defined the fidelity $\lambda$ to be:

$$\lambda = e^{-\frac{D^2}{T_D^2}}$$

(2.30)

where $T_D$ is a user-settable parameter that represents the allowed deviation from the observed data and $D$ is assumed to have a Gaussian distribution. This is inspired in the Kolmogorov-Smirnoff test (eq. 2.31),

$$D = \frac{1}{|I|} \int_I \left| \hat{F}(x) - F_n(x) \right| dx,$$

(2.31)

where $F_n(x)$ is the empirical cumulative distribution of the $n$ observations, $\hat{F}(x)$ the corresponding cumulative Gaussian distribution, and $I$ the interval within which the two functions are compared. While the Kolmogorov-Smirnoff test grows unbounded with $n$, the fidelity $\lambda$ provides a bounded quantification of the correspondence between the model and the data. The decision to merge two Gaussians $G_i$ and $G_j$ is only made if the resulting Gaussian has a $\lambda$ value close to 1, by exceeding a given threshold, $\lambda_{\min}^+ = 0.95$ for example. The fused Gaussians are computed by the combination of the cumulative density functions:

$$\pi = \pi_i + \pi_j,$$

(2.32)

$$\mu = \frac{1}{\pi} [\pi_i \mu_i + \pi_j \mu_j],$$

(2.33)

$$C = \left( \frac{\pi_i}{\pi} \left[ C_i + (\mu_i - \mu) \top (\mu_i - \mu) \right] \right) + \left( \frac{\pi_j}{\pi} \left[ C_j + (\mu_j - \mu) \top (\mu_j - \mu) \right] \right),$$

(2.34)
\[ F(x) = \frac{1}{\pi} \left[ \pi_i F_i(x) + \pi_j F_j(x) \right], \quad (2.35) \]

When a new data point comes in, an attempt is made to merge the corresponding Gaussian, i.e., the Gaussian generated to represent that point, with its nearest Gaussian already present in the mixture. Only try the nearest Gaussian is tried, i.e., the nearest neighbor, because it is the most likely to provide a precise simplification. If the two Gaussians are successfully merged, the resulting Gaussian is again a candidate for simplification. Merging continues until the best candidate to merge drops below \( \lambda_{\text{min}}^+ \). This is a very fast algorithm, as it corresponds to two nested loops containing one nearest neighbor search and one merge, respectively. The low computational cost of this merge approach makes it suitable for online applications.

The problem with this approach is that it leads to gross overfitting of sparse data. Consider the scenario where we have a sparse distribution of data. In the beginning, each data point increases the complexity by one, but as more points are added, the simplification step takes effect, and the complexity decreases until it converges to the tradeoff between complexity and accuracy defined by \( T_D \) in eq. 2.30. In the sparse data scenario, this results in a single Gaussian with a large covariance. As more data comes in, all new incoming data points will be merged with the broad Gaussian because it is enough to explain the data above the threshold \( \lambda_{\text{min}}^+ \). This leads to a broad overfitting of the data.

The second level deals with splitting the known Gaussians. A new threshold \( \lambda_{\text{min}}^- \) is introduced.

\[ \lambda_{\text{min}}^- = \exp \left( \frac{N \log \lambda_{\text{min}}^+}{\chi^2_{N-1}(\alpha)} \right), \quad (2.36) \]

where \( \alpha \) is the probability of incorrectly splitting the Gaussian. A \( \chi^2 \) distribution is used, since \( T_D \) represents a standard deviation, and empirical estimates of variance follow a \( \chi^2 \) distribution. Using this threshold and the uncertain Gaussian model, Declercq and Piater (2007) are able to learn a GMM that is kept as general as possible until there is enough evidence to make it more specific.

The uncertain Gaussian model represents a distribution with an appropriately weighted sum of informative (Gaussian) and uninformative (uniform) components,

\[ q(x) = \lambda \exp \left( -\frac{1}{2} (x - \mu)^\top \tilde{C}^{-1} (x - \mu) \right) + (1 - \lambda), \quad (2.37) \]

where \( \tilde{C} \) is an augmented covariance that bounds the risk of underestimating the true covariance, i.e., \( P(\tilde{C} \leq C) = \alpha \) where conventionally \( \alpha = 0.05 \). The drawback is that, when they want to split a Gaussian, the observations are not in memory anymore, leaving them with a poor model that can no longer be refined. This is the source for the motivation for the two-level mixture. Data is represented by the uncertain mixture, and each uncertain Gaussian component contains a precise mixture to describe itself. Thus, when a split of an uncertain Gaussian is needed, we can split it according to its underlying mixture components. The algorithm is summarized as follows:

1. Merge the new data point with the nearest uncertain Gaussian

2. If the resulting Gaussian has \( \lambda < \lambda_{\text{min}}^- \), replace it with two Gaussians learned from their underlying GMM with EM
3. else continue to merge the current uncertain Gaussian with its nearest neighbor until the resulting Gaussian has $\lambda < \lambda_{\text{min}}$.

Note that merging two uncertain Gaussians also involves merging their respective underlying mixture models. This can be done by summing the components from both mixtures and using the simplification step only on the precise Gaussian that contains the new observation.

2.5 Summary

In this chapter we presented related work regarding Gaussian Mixture Models. The theoretical model and traditional approach to batch estimation, using the Expectation Maximization algorithm, was first presented. The limitations of EM were also addressed. Kernel Density Estimation was then presented an alternative non-parametric approach to GMM estimation which did not suffer from some of the previous limitations. Addressing our concerns regarding online operation, several approaches of online GMM estimation were presented. This description includes two works from Kristan, Skočaj, and Leonardis (2010) and Declercq and Piater (2007) that serve as basis of the state of the art work presented in the next chapter.
In an incremental learning environment, samples are not available all at once, and we can have a potentially very large number of samples. In this sense, it is not computationally feasible to maintain all data points. Typical approaches to kernel estimation rely on having access to the complete sample set. Other approaches, which deal with online operation, make assumptions about data distributions. This has the advantage of better constraining the problem and designing efficient algorithms for the target task. However, when these assumptions are violated, these algorithms may converge to poor solutions or even fail to converge.

When trying to design a generic algorithm, one must make the minimum possible assumptions about data distribution and other working constraints, such as sample dimensionality or even the number of samples to be provided. For this, we maintain a non-parametric model of the data in a form of a sample distribution. This model can then be used to calculate the kernel density estimate of the target distribution.

Recently Kristan, Leonardis, and Skočaj (2011) proposed an approach to online kernel density estimation that fulfills most of our constraints with a solid and adaptable approach. This chapter will focus on describing and discussing this approach. An even more recent work from the same authors is available (Kristan and Leonardis 2014), which is based on the previous work, but focused on building discriminative classifiers. Since the base approach is the same, we decided to follow both papers when complementary information was available, but ignoring the classifier building adaptation of the recent paper, as it is not our main task.

During online operation, each new observation is viewed as a Dirac-delta function and the current sample model is updated accordingly. However, if nothing else is done, the number of components would increase linearly with the observed data. This would defeat the initial objective of having a model representing the observed samples without having to maintain all samples. To avoid this we can compress the existing model by fitting several samples using a single component. This allows representing a similar amount of information without keeping all samples. However, the model is not stationary, and a compression which made sense at some point in time, may no longer make sense at a later time. To allow recovery from these over-compressions, each component of the sample distribution keeps a detailed model composed of two Gaussians. Note that choosing to keep only two Gaussians allows to have the simplest model that allows detection and recovery from over-compressions.

The authors aim at maintaining a (possibly compressed) distribution model of the observed samples and use it to calculate the KDE when needed. Each sample is simply a single D-dimensional data-point and can be represented as a Dirac-delta function with its probability mass concentrated at that data-point. We can generally write this as a Gaussian probability density function (pdf) with zero covariance.

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

\[
p_s(x) = \sum_{i=1}^{N} w_i \phi_{\Sigma_i}(x - x_i)
\]  

(3.1)

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

\[
\phi_{\Sigma_i}(x - \mu_i) = \frac{1}{2\pi^{D/2}|\Sigma_i|^{1/2}} \exp \left\{ \frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) \right\},
\]

(3.2)

with mean vector \(\mu_i\) and covariance matrix \(\Sigma_i\). The sum of the mixture weights is constrained such that \(\sum_{i=1}^{N} w_i = 1\).

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

\[
S_{\text{model}} = \{p_s(x), \{q_i(x)\}_{i=1:N}\}
\]

(3.3)

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

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

\[
\hat{p}_{\text{KDE}}(x) = \phi_{H}(x) * p_s(x) = \sum_{i=1}^{M} w_i \phi_{H+\Sigma_i}(x - x_i)
\]

(3.4)

### 3.1 Approach Overview

Now that the basic model is defined, and before presenting the details of each component of the online kernel density estimation approach, it is advisable to have an intuitive grasp of the basic ideas involved. The whole approach is based on three concepts: optimal bandwidth, compression, and revitalization.

Each new observation is added to the model in the form of a Dirac-delta function, i.e., a Gaussian with mean \(x\) and variance \(\Sigma=0\). As more samples are added, 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 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 3.1 presents a graphical overview of the process.
Figure 3.1: Global overview of a full iteration of the oKDE approach (a). The sample model $S_{t-1}$ is updated by a new observation $z_t$ 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 (Kristan, Leonardis, and Skočaj 2011).

The following sections describe these three central concepts in detail. Section 3.2 describes the optimal bandwidth estimation process. Section 3.3 describes the revitalization and the compression approach, formulating a measure of distance between mixtures of Gaussians in section 3.3.1 and describing an hierarchical approach to compress the mixture model in section 3.3.2.

Section 3.4 presents auxiliary algorithms that are part of the oKDE approach but that could be replaced by another equivalent solution. This section, first, provides a detailed description of the distance between mixtures, as defined by the unscented Hellinger distance in section 3.4.1. The clustering of Gaussian kernels approach proposed by Goldberger and Roweis (2004) is then presented in section 3.4.2.

To finalize, an analysis of the algorithm and its existing implementation weaknesses is presented in section 3.5. This section describes a few critical aspects, namely the numeric problems of stability for arbitrary data and high dimensionality, and computational performance of the currently available implementation. This discussion forms the basis for the new implementation and our replication of the original oKDE approach in a different programming language and taking the identified problems into consideration.

### 3.2 Optimal Bandwidth Estimation

The goal of all KDE methods is to determine the kernel bandwidth $H$ such that the distance between the $\hat{p}_{\text{KDE}}(x)$ and the unknown pdf $p(x)$ that generated the data, is minimized. When $p(x)$ is unknown, as in our case, a classical measure of closeness of $\hat{p}_{\text{KDE}}(x)$ to the underlying pdf is the asymptotic mean integrated squared error (AMISE) (Wand and Jones 1994), defined as

$$AMISE = (4\pi)^{-d/2}|H|^{-1/2}N^{-1} + \frac{1}{4}d^2 \int \text{tr}^2 \{H \mathcal{G}_p(x)\} dx \quad (3.5)$$

where $\text{tr}\{\cdot\}$ is the trace operator, $\mathcal{G}_p$ is the Hessian of $p(x)$, and $N$ is the number of components of the mixture. Rewriting the bandwidth matrix in terms of scale $\beta$ and structure $F$,

$$H = \beta^2 F \quad (3.6)$$
CHAPTER 3. ONLINE KERNEL DENSITY ESTIMATION

Assuming that the structure of \( \mathbf{F} \) is known, then eq. 3.5 is minimized at scale

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

(3.7)

where the term

\[
R(p, F) = \int \text{tr}^2 \left\{ F G_p(x) \right\} dx
\]

(3.8)

is a functional of the second-order partial derivatives, \( G_p(x) \), of the unknown distribution \( p(x) \). This functional could be estimated using the plug-in methods. However, these methods assume access to all previously observed samples, which, in our setup, is not true. Since only a compressed mixture model of the samples is maintained, an approximation to the functional using this mixture model is required. First, note that \( R(p, F) \) can be written in terms of expectations of the derivatives

\[
\psi_r = \int p^r(x) p^r(x) p(x) dx
\]

(3.9)

The sample distribution \( p(x) \) can thus be used to make the following approximations:

\[
p(x) \approx p_s(x), \quad p^r(x) \approx p^r_G(x)
\]

(3.10)

The derivative of \( p(x) \), \( p^r_G(x) \), can be approximated by the kernel density estimate

\[
p^r_G(x) = \phi_G * p_s(x) = \sum_{j=1}^N w_j \phi_{\Sigma_{ij}}(x - x_j)
\]

(3.11)

The estimate \( p^r_G(x) \) plays the role of the so-called pilot distribution with covariance terms

\[
\Sigma_{ij} = G + \Sigma_{sj}
\]

(3.12)

where \( \Sigma_{sj} \) is the covariance of the \( j \)th component of \( p_s(x) \) and \( G \) is called the pilot bandwidth, which is estimated by a multivariate normal-scale rule of the distributions derivative

\[
G = \hat{\Sigma}_{\text{mp}} \left( \frac{4}{d+2} N \right)^{2/(d+4)}
\]

(3.13)

Using the approximations in eq. 3.10 we can approximate \( R(p, F) \) by

\[
\hat{R}(p, F, G) = \int \text{tr} \{ F G_{p_G}(x) \} \text{tr} \{ F G_p(x) \} dx
\]

(3.14)

Since \( p_s(x) \) and \( p^r_G(x) \) are both Gaussian mixture models, the functional 3.14 can be computed using only matrix algebra:

\[
\hat{R}(p, F, G) = \sum_{i=1}^N \sum_{j=1}^N w_i w_j \phi^{-1}_{A_{ij}}(\Delta_{ij}) \times [2 \text{tr}(F^2 A_{ij}^2)] [1 - 2m_{ij}] + \text{tr}^2(F A_{ij}) [1 - m_{ij}]^2]
\]

(3.15)

where the following definitions were used:

\[
A_{ij} = (\Sigma_{ij} + \Sigma_{sj})^{-1}, \quad \Delta_{ij} = \mu_i - \mu_j, \quad m_{ij} = \Delta_{ij}^T A_{ij} \Delta_{ij}
\]

(3.16)

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 (Wand and Jones 1994; Duong and Hazelton 2003), then \( F = \hat{\Sigma}_{\text{mp}} \). Note that, if data has been spherized (Silverman 1986) the bandwidth structure \( F \) will correspond to the identity matrix, \( I \), and \( F \) could simply be removed from equations 3.6, 3.12, and 3.15.
3.3 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 is to approximate the original $N$-component sample distribution

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

(3.17)

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

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

(3.18)

such that the resulting compressed KDE does not change significantly. This means that the same information is being approximately represented using fewer components, while minimizing the induced error.

Since the reference mixture $p_s(x)$ is known, the difference between the reference mixture and its approximation can be quantified by the integrated squared error (ISE) (eq. 2.25) and the problem of finding an equivalent $\hat{p}_s(x)$ can be posed as seeking an optimal set of mixture parameters $\theta = \{ w_j, \mu_j, \Sigma_j \}_{j=1:M}$, while minimising the ISE. This direct optimization (Kristan, Skočaj, and Leonardis 2010), however, can be computationally prohibitive, and prone to slow convergence even for a moderate number of dimensions. In a high dimensionality scenario, this approach is even less appropriate.

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

$$p_s(x; \pi_j) = \sum_{i \in \pi_j} w_i \phi_{\Sigma_i}(x - x_i)$$

(3.19)

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

$$\hat{w}_j = \sum_{i \in \pi_j} w_i, \quad \hat{\mu}_j = \hat{w}_j^{-1} \sum_{i \in \pi_j} w_i \mu_i, \quad \hat{\Sigma}_j = \left( \hat{w}_j^{-1} \sum_{i \in \pi_j} w_i (\Sigma_i + \mu_i \mu_i^T) \right) - \hat{\mu}_j \hat{\mu}_j^T$$

(3.20)

In figure 3.2, an example is provided where the components of a sample distribution $p_s(x)$ are clustered to form another (compressed) sample distribution $\hat{p}_s(x)$. 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.

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,
which components of a sample distribution

\[ \text{sub-mixture:} \]

the first two moments (mean and covariance)

\[ \text{clusters} \]

under the KDE with bandwidth

\[ \text{respectively.} \]

i.e., it does not exceed a predefined threshold \( D_{th} \)

\[ \hat{M} = \arg \min_{M} E(\Xi(M)) \quad \text{s.t.} \quad E(\Xi(\hat{M})) \leq D_{th} \]  

(3.21)

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

\[ E(\Xi(M)) = \max_{\pi_j \in \Xi(M)} \hat{E}(p_{s}(x; \pi_j), \mathbf{H}_{\text{opt}}) \]  

(3.22)

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

### 3.3.1 Local Clustering Error

Consider \( p_{1}(x) \) as a cluster, i.e., a sub-mixture of the sample distribution, as shown in eq. 3.19. Also, consider \( p_{0}(x) \) to be the resulting single Gaussian approximation described in 3.20. And finally, let \( \mathbf{H}_{\text{opt}} \) to be the current estimation of the optimal bandwidth. The local clustering error is defined as the distance

\[ \hat{E}(p_{1}(x), \mathbf{H}_{\text{opt}}) = D(p_{1\text{KDE}}(x), p_{0\text{KDE}}(x)) \]  

(3.23)

between the corresponding KDEs

\[ p_{1\text{KDE}}(x) = p_{1}(x) * \phi_{\mathbf{H}_{\text{opt}}}(x) \]

\[ p_{0\text{KDE}}(x) = p_{0}(x) * \phi_{\mathbf{H}_{\text{opt}}}(x) \]

(3.24)

One way to quantify the distance between distributions is using the Hellinger distance (Pollard 2002), which is defined as

\[ D^2(p_{1\text{KDE}}(x), p_{0\text{KDE}}(x)) \triangleq \frac{1}{2} \int \left( (p_{1\text{KDE}}(x))^{1/2} - (p_{0\text{KDE}}(x))^{1/2} \right)^2 dx \]  

(3.25)

A formal description of this distance is provided in section 3.4.1, later in this chapter.
3.3. MODEL COMPRESSION

3.3.2 Hierarchical Compression

The global optimization of eq. 3.21 would require the evaluation of all possible cluster assignments \( \Xi(M) \). If we view the components of the sample model \( p_s(x) \) as elements of a set. The total number of possible cluster assignments can be represented the Bell number (Bell 1938) \( B_N \), where \( N \) is the number of components. The first 12 Bell numbers are

\[
\{ 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570 \}
\]

A more exhaustive list is available in (Sloane 2011), but the point is that this number grows rapidly. If we have a conservative number of component, for example, 30, the corresponding Bell number is \( B_{30} = 8.467 \times 10^{23} \). This makes it very clear that a brute force evaluation of all possible assignments is computationally prohibitive. A significant reduction in complexity can be achieved by a hierarchical approach to clustering. Similar approaches have been successfully applied for data compression of Gaussian mixture models to a predefined number of components (Goldberger and Roweis 2004; Ihler 2005).

The hierarchical compression algorithm starts by splitting the entire sample distribution \( p_s(x) \) into two sub-mixtures using Goldberger’s K-means algorithm (Goldberger and Roweis 2004) for mixture models, with \( K=2 \). In order to avoid singularities associated with Dirac-delta components of the sample model, instead of \( p_s(x) \), the K-means algorithm is applied to the corresponding KDE model \( p_{KDE}(x) \). Next, each sub-mixture is approximated by a single Gaussian using eq. 3.20 and the sub-mixture which yields the largest local error \( \hat{E}(p_1(x), H_{opt}) \) is further split into two sub-mixtures. This process continues until the largest local error is sufficiently small to fulfill the conditions \( E(\Xi(M)) \leq D_{th} \) in eq. 3.21. This process produces a binary tree with \( M \) leaves, where each leaf represents the clustering assignments \( \Xi(M) = \{ \pi_i \}_{j=1:M} \) between the components of the sample distribution \( p_s(x) \). The leaves also provide the single Gaussian approximation given by eq. 3.20 of the clustering assignment. However, since we applied this approach to the KDE model we cannot directly use the leaves to produce the compressed sample distribution \( \hat{p}_s(x) \) (eq. 3.18). We therefore use the single Gaussian approximation in the sample distribution \( p_s(x) \) to correspond to each component in the compressed sample distribution \( \hat{p}_s(x) \). Figure 3.3, illustrates this process.

![Hierarchical Clustering example](image)

Recall that we keep track of a detailed model for each component in the sample distribution. This means that, if two components are merged, their underlying detailed models should
be merged as well. The detailed model $\hat{q}_j(x)$ of the $j$-th component in the compressed model $\hat{p}_s(x)$ is calculated by first defining a normalized extended mixture model.

$$\hat{q}_{jext}(x) = \left( \sum_{i \in \pi_j} w_i \right)^{-1} \sum_{i \in \pi_j} w_i q_i(x) \quad (3.27)$$

If the extended mixture has one or two components, then $\hat{q}_j(x) = \hat{q}_{jext}(x)$. If this mixture has more than two components, then the two-component detailed model $\hat{q}_j(x)$ is generated by splitting $\hat{q}_{jext}(x)$ into two sub-mixtures using Golberger K-means and approximating each sub-mixture by a single Gaussian using eq. (3.20).

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.

### 3.3.3 Revitalization

The compression algorithm described before allows to identify and compress clusters of components whose compression do not introduce a significant error into the KDE with bandwidth $H_{opt}$ estimated at the time of compression. However, as more samples arrive, the sample distribution changes. Consequently, both $H_{opt}$ and KDE 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 are detected through the inspection of the detailed model of each component in the sample distribution $p_s(x)$. The local clustering error $\hat{E}(q_i(x), H_{opt})$ (3.22) of each component in the sample distribution is evaluated against its detailed model $q_i(x)$ to test whether the error exceeds the threshold $D_{th}$. Those components in $p_s(x)$ for which $\hat{E}(q_i(x), H_{opt}) > D_{th}$, are removed from the sample distribution, being replaced by the two components of their detailed model. These new components, however, need their own detailed model. For this, a new detailed model is generated, based on the covariances of each of the new components. For example, let $w_i \phi_{\Sigma_i}(x - \mu_i)$ be one of the new components. If the component is a Dirac-delta function, corresponding to a single data-point, then its detailed model is just the component itself. However, if the component has been generated through clustering of several detailed models in previous compression steps, then it needs the two-component mixture detailed model. This detailed model is initialized by splitting $\phi_{\Sigma_i}(x - \mu_i)$ along its principal axis (Huber 2015) 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 $UDU^T = \Sigma_i$ be a singular value decomposition of $\Sigma_i$ with singular values and singular vectors ordered by decreasing singular values. Then, the component detailed mixture
model is defined as
\[ q_i(x) = \sum_{k=1}^{2} w_k \phi_{\Sigma_k}(x - \mu_k), \]
\[ \mu_1 = FM + \mu_i, \quad \mu_2 = FM - \mu_i, \]  \hspace{1cm} (3.28)
\[ \Sigma_k = FCF^T, \quad w_k = \frac{1}{2} w_i \]

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

### 3.4 Secondary Algorithms

The previous sections described the main ideas and elements of the oKDE approach. However, for some of those elements, we can separate the idea from the actual implementation. One example could be the compression algorithm, for which the Golberger K-means was chosen, but could be replaced by another, as long as, in the end, provides a compressed version of the sample model. The following sections will describe these secondary, i.e., replaceable, algorithms. We start by describing the Hellinger distance in section 3.4.1, which defines the distance between two mixtures of Gaussians. Then, we describe our implementation of the Golberger K-means in section 3.4.2.

#### 3.4.1 Hellinger Distance

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 (Julier and Uhlmann 1996), 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 (Veach and Guibas 1995), an importance distribution is defined,

\[ p_0(x) = \gamma(p_1(x) + p_2(x)) \]  \hspace{1cm} (3.29)

which contains the support of both \( p_1(x) \) and \( p_2(x) \), with \( \gamma \) set such that \( \int p_0(x)dx = 1 \). In our case \( p_0(x) \) is a Gaussian mixture model, so equation \eqref{eq:importance} becomes

\[ p_0(x) = \sum_{i=1}^{N} w_i \phi_{\Sigma_i}(x - x_i) \]  \hspace{1cm} (3.30)

We can then rewrite the Hellinger distance (eq.3.25) as

\[ D^2(p_1, p_2) = \frac{1}{2} \int g(x)p_0(x)dx = \frac{1}{2} \sum_{i=1}^{N} w_i \int g(x)\phi_{\Sigma_i}(x - x_i)dx \]  \hspace{1cm} (3.31)

where

\[ g(x) = \left( \frac{\sqrt{p_1(x)} - \sqrt{p_2(x)}}{p_0(x)} \right)^2 \]  \hspace{1cm} (3.32)
CHAPTER 3. ONLINE KERNEL DENSITY ESTIMATION

Note that the integrals in eq. 3.31 are expectations over a nonlinearily transformed Gaussian random variable $X$, and therefore admit to the unscented transform. According to Julier and Uhlmann (1996) we can approximate $D^2(p_1, p_2)$ according to eq 3.33

$$D^2(p_1, p_2) \approx \frac{1}{2} \sum_{i=1}^{N} w_i \sum_{j=0}^{2d+1} g^{(j)}(i) \mathcal{W}_i$$  \hspace{1cm} (3.33)

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

$$^{(0)}X_i = x_i, \hspace{1cm} ^{(0)}\mathcal{W}_i = \frac{k}{d+k}$$

$$^{(j)}X_i = x_i + s_j \sqrt{(d+k)\xi_j}, \hspace{1cm} ^{(j)}\mathcal{W}_i = \frac{1}{2(d+k)}$$  \hspace{1cm} (3.34)

where $k = \max([0, m-d])$ and $\xi_j$ is the $j$-th column of the matrix square root of $\Sigma_i$ such that $\xi = \sqrt{\Sigma_i}$. Note that $j$ was defined in the range $[0; 2d+1]$, and the square matrix $\xi$ contains only $d$ columns. This apparent notation confusion is cleared if we take into account that sigma points are simply the mean $x_i$, and $x_i \pm \xi_j$, i.e., the sum and subtraction of each of the dimensions of the covariance matrix $\Sigma_i$. Therefore, the $j$-th column of $\xi_j$ 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 $UDU^T$ be a singular value decomposition of covariance matrix $\Sigma$, such that $U = \{U_1, \ldots, U_d\}$ and $D = \text{diag}(\lambda_1, \ldots, \lambda_d)$, 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 not be positive semi-definite. In line with these discussions, presented by Julier and Uhlmann (1996), we set the parameter $m$ to $m = 3$ and force a non-negative value $k$ resulting in $k = \max([0, 3-d])$.

3.4.2 Goldberger K-Means

The previous hierarchical compression and revitalization algorithms rely on the clustering of Gaussian distributions. One way to do this is by using the Goldberger K-means (Goldberger and Roweis 2004) for mixture models.

Suppose we have an $N$-component Gaussian mixture

$$f(x) = \sum_{i=1}^{N} w_i \phi_{\Sigma_i}(x - x_i) = \sum_{i=1}^{N} w_i f_i(x)$$  \hspace{1cm} (3.35)

and want to cluster the components of $f$ into a reduced mixture of $K < N$ components. Considering $g$ to be the the mixture of size $K$ that we wish to find, then the goal is to find the mixture $g$ that is the closest to $f$ under some distance measure. The distance measure between
3.4. SECONDARY ALGORITHMS

\[ f = \sum_{i=1}^{N} w_i f_i \text{ and } g = \sum_{j=1}^{K} \beta_j g_j \text{ can be defined as} \]

\[ d(f, g) = \sum_{i=1}^{N} w_i \min_{j=1}^{K} KL(f_i || g_j) \] (3.36)

which can be thought of as the cost of coding data generated by \( f \) under model \( g \), if all points generated by component \( i \) of \( f \) are coded under a single component of \( g \). This distance can be analytically computed, as each term is the Kullback-Leibler divergence (Kullback 1968) between two Gaussian distributions \( f_j = \phi_{\Sigma_1}(x - \mu_1) \) and \( g_j = \phi_{\Sigma_2}(x - \mu_2) \)

\[ KL(f_j || g_j) = \frac{1}{2} \left( \log \frac{|\Sigma_2|}{|\Sigma_1|} + tr(\Sigma_2^{-1}\Sigma_1) + (\mu_1 - \mu_2)^T \Sigma_2^{-1}(\mu_1 - \mu_2) - d \right) \] (3.37)

where \( tr \) is the trace operator and \( d \) is the number of dimensions. Under this divergence, the optimal reduced mixture \( \hat{g} \) is the solution to the minimization of eq. 3.36 over \( g \)

\[ \hat{g} = \arg \min_g d(f, g) \] (3.38)

Goldberger and Roweis (2004) prove that the optimal solution \( \hat{g} \) is a Gaussian mixture obtained by grouping the components of \( f \) into clusters and collapsing all Gaussians within a cluster into a single Gaussian. Since there is no closed-form solution for the minimization, the authors resort to an iterative algorithm to obtain a locally optimal solution. Let \( S \) define the set of all mappings from \( \{1, \ldots, N\} \) to \( \{1, \ldots, K\} \). For each mapping \( \pi \in S \) we can define

\[ d(f, g, \pi) = \sum_{i=1}^{N} w_i KL(f_i || g_{\pi_i}) \] (3.39)

Now let \( \pi_i \in S \) be a matching function

\[ \pi_i = \arg \min_{j=1}^{K} KL(f_i || g_j) \quad i = 1, \ldots, N \] (3.40)

where the goal of is to find the mappings that minimize the KL-divergence between each \( f_i \) and \( g_j \). This gives us the optimal mapping between \( f_i \) and \( g_j \) and we can use it to define our main optimization:

\[ d(f, g) = d(f, g, \pi) = \min_{\pi \in S} d(f, g, \pi) \] (3.41)

Using equations 3.36 and 3.41, the optimal reduced model can be found by the solution to the following double minimization problem:

\[ \hat{g} = \arg \min_g \min_{\pi \in S} d(f, g, \pi) \] (3.42)

Let \( f_{\pi_j} \) be the sub-mixture corresponding to the set \( \pi_i \) components, defined by

\[ f_{\pi_j} = \sum_{i \in \pi_j} w_i f_i \]

\[ \sum_{i \in \pi_j} w_i \]

Let \( g_j \) be defined as the one Gaussian approximation of \( f_{\pi_j} \). For this we can use the equations described in eq. 3.20 which are re-written here in a slightly different version for increased clarity

\[ \beta_j = \sum_{i \in \pi_j} w_i, \quad \mu_j = \beta_j^{-1} \sum_{i \in \pi_j} w_i \mu_i, \quad \Sigma_j = \left( \beta_j^{-1} \sum_{i \in \pi_j} w_i (\Sigma_i + \mu_i \mu_i^T) \right) - \mu_j \mu_j^T \] (3.44)
which satisfies

\[ g_j = \phi_{\Sigma_j}(x - \hat{\mu}_j) = \arg \min_g KL(f_j || g) = \arg \min_g d(f_j, g) \]  

(3.45)

This means that \( g_j \) is a Gaussian distribution that represents the set \( \pi_i \) that minimizes the distance between both distributions. The collapsed version of \( f \) can, thus, be defined as

\[ g = \sum_{j=1}^{N} \beta_j g_j \]  

(3.46)

The iterative minimization algorithm can be viewed as a type of K-means operating at the meta-level of model parameters

\[
\begin{align*}
\pi &= \arg \min_{\pi} d(f, g, \pi) \quad \text{(REGROUP)} \\
g &= \arg \min_{g} d(f, g, \pi) \quad \text{(REFIT)}
\end{align*}
\]

where \( \pi_i = \arg \min_j KL(f_i || g_j) \) and \( g \) is computed using eq. 3.46. In other words, what happens is that in, the (REGROUP) phase, we find the mappings from each \( f_i \) to a \( g_j \) that minimizes the distance \( d(f, g) = KL(f_i || g_j) \). And in the (REFIT) phase, we regroup those mappings into single Gaussian approximations, resulting in a new \( g \). This iterative process monotonically decreases the distance measure \( d(f, g) \) and, since \( S \) is finite, the algorithm converges to a local minimum after a finite number of iterations.

This completes the formal description of the Goldberger K-Means algorithm. This, however, leaves some ambiguity on how to solve certain issues that surely arise on an actual implementation of the algorithm. In the following subsections, we will cover some of those issues, namely initialization, under-mapping and convergence strategies.

### 3.4.2.1 Initialization

As with all K-means approaches, the initialization of the algorithm can strongly affect the final result. One typical approach is to randomly select \( K \) points as the starting pivots of the algorithm. Since we are working with Gaussian distributions, we would either have to randomly generate \( K \) Gaussians, including the corresponding mean and covariances, or, as a potentially better solution, randomly select \( K \) Gaussians from the mixture we want to compress as our pivots. This has the usual disadvantage that the pivots may be too close to each other and skew the final result for one part of the whole Gaussian mixture. One way to try to mitigate this issue is to take into consideration the weights of the Gaussians and select the \( K \) highest weighting as our starting pivots. This has the advantage of preserving the relative importance of the Gaussians in the mixture, which means that our initial pivots will correspond to the densest areas of the model. However, this too has the disadvantage of not preserving the complete structure of the model, as there may be cases were the highest density resides in a particular part of the model.

A third solution, which was adopted in our implementation, is applicable when \( K=2 \), which is our case. Here, we can approximate the whole Gaussian mixture by a single Gaussian using eq. 3.20 and then split that Gaussian in two along its principal axis by using eq. 3.28. For higher values of \( K \), similar solutions may be employed but will not be covered as our current goal is to have a K-means with \( K=2 \) clusters.
3.4. SECONDARY ALGORITHMS

3.4.2.2 Under-mapping

During execution of the algorithm there can be situations where all components are assigned to a single pivot, leaving the other with no assignments. If nothing else was done, and the algorithm continued, one of the new pivots would be empty, i.e., undefined. This breaks the algorithm, as the output is no longer composed of two clusters of Gaussians but rather a single cluster containing all Gaussians.

This issue can be solved by selecting the component which yields the highest KL-divergence to the assigned pivot and assign it to the other (empty) pivot. This results in a cluster containing $K - 1$ components and another containing only one component. This allows the computation of the new pivots for the next iteration, where one is simply the assigned component, and the other is the approximation of the remaining components by a single Gaussian.

3.4.2.3 Convergence

The convergence test proposed by Goldberger and Roweis (2004) involved computing the KL-divergence of each component $f_i$ relative to the new corresponding pivot $g_i$. In other words, we measure the divergence of the components in the sets $\pi_i$, $i=1:K$ to the corresponding one Gaussian approximation $g_i$. This global error allows us to test for convergence in two ways.

The first approach, is to first check if we are below a desired threshold, set according to the maximum allowed global divergence error. If it is below, we can then test if this error is below some threshold between this iteration global divergence and the previous one, indicating us whether or not the algorithm is converging too slowly and should be stopped. Three problems arise from this approach. First, it does not take into account possible cycles, where a new iteration assignments are equal to a previous one, causing the algorithm to enter a loop stopped only when some maximum iteration threshold is reached. Second, it involves computing the KL-divergence for each component, which impacts performance, especially when we have a potentially large number of components. Third, this involves defining two thresholds, which would have to be tuned depending on the task. This is a non-ideal situation as we are trying to build a non-parametric model of the data providing the least possible amount of prior knowledge.

An alternative approach, which we used in our implementation, is to save the mapping vector in each iteration and then use it to check for convergence and cycles. If the current iteration mapping vector is equal to the one from the previous iteration, it means we have achieved convergence. If it is equal to a previous iteration, then we are in the presence of a cycle and should stop the algorithm. To take into account the possibility of low convergence speed we also set a maximum iteration threshold of 20, as the the algorithm typically converges faster.

3.4.2.4 Goldberger K-means Algorithm

The previous sections presented the algorithm in a formal language and discussed issues that were not considered in the original paper (Goldberger and Roweis 2004). Implementation decisions and their motivation were also described. Our implementation is presented in full in Algorithm 2.
input: A Gaussian Mixture Model with N components
output: A Gaussian Mixture Model with K=2 components
Approximate input GMM by a single Gaussian using (3.44);
Split this Gaussian in two by using (3.28);
Use each of the two parts of the split to set pivots Piv1 and Piv2;
Set iteration (it) = 0;
Set mapping[it] = zeros(N) - vector of size N containing all zeros;
while true do
  for each GMM component (cp) do
    if $\text{KL}(cp \parallel Piv1) < \text{KL}(cp \parallel Piv2)$ then
      mapping[it][cp] = Piv1;
    else
      mapping[it][cp] = Piv2;
    end
  end
  for each Piv do
    Piv = approximation of all respectively mapped components using eq. 3.44;
  end
  if mapping[it] equal to any of previous mapping then
    stop
  end
  else if $it \geq \text{maxIterations}$ then
    stop
  end
end

Algorithm 2: Goldberger K-means
3.5 Critical Analysis

Online kernel density estimation approach fulfills our goal of modeling data through time, while allowing it to change and evolve without the need to keep all previous observations in memory. However, it presents a few weaknesses regarding high dimensionality and numerical stability.

One of the elements of the Gaussian likelihood function is computing the determinant of the covariance matrix. However, for high dimensional data this matrix will be large. If data variance is very high, or very low, there may be situations where the determinant causes an overflow, or underflow, for a given numerical precision representation.

Since we aim to model data with arbitrary dimensionality and distribution, we need to ensure that this approach is capable of representing irregular data, with skewed distribution and near immutable features, i.e., features that seldom change. This scenario may originate situations where the covariance of the entire model, needed for bandwidth computation, is rank-deficient, and thus non-invertible, which may cause convergence issues, especially for the Dirac-delta functions present in the model. Similarly, data distribution may produce optimal bandwidths so close to zero that surpass available precision, which implies that Dirac-deltas would remain without any covariance.

Furthermore, the originally available implementation of this approach (Kristan, Leonardis, and Skočaj 2011) is research MATLAB code, not-optimized for speed and highly redundant in terms of code. It does not take into account the high dimensionality scenario, and although it has rank correction approaches for covariance matrices it has no strategy for correcting the optimal bandwidth. This is acceptable if the datasets have been subject to a prior normalization, such as the features having unitary variance. But we do not want, and sometimes cannot, explicitly perform any normalization, as we assume data distribution may change as more samples arrive. Besides, pre-normalizing means we need complete access to the data samples. In our setup, we assume that these samples are not available beforehand and are fed to the model directly as they are being captured.

Either due to numeric issues or to time and memory constraints, the current implementation was not able to cope with some datasets. We could adapt the current implementation to increase numerical stability, but the time and memory inefficiency would remain. Plus, since the code is highly redundant, changing it is prone to the introduction of bugs making the effort even higher for the little to moderate benefit we would obtain.

A more expensive approach in the short term, but with higher gain in the long term is to completely re-implement the whole method, and taking into account all the issues raised in this section. This allows to leverage the benefits of using object oriented programming and increase computational performance at the same time. While the basic implementation follows this chapter algorithm we introduced several numeric stability and performance improvements. These are described in the next chapter.

3.6 Summary

In this section, oKDE (Kristan, Leonardis, and Skočaj 2011), an online kernel density estimation approach that fulfills our initial requirements was presented in detail. An overview and
detailed mathematical description of the approach was presented, showing how the model evolves by splitting and merging as more samples are added: each incoming sample is viewed as a new component in the mixture, represented by a Dirac-delta function. As more samples are added, the compression threshold, which controls when the compression routine is called, may be exceeded, causing the compression step to be executed. This step consists of two phases: revitalization and hierarchical compression. In the first phase, each component is evaluated to assess whether it is still a good approximation of its own underlying model. If it deviates too much from its internal model, then it is split into two new components. For each new component, a new underlying model is also computed. This first phase ensures that all components in the model are good representations of the observed samples. In the second phase (hierarchical clustering), the goal is to reduce redundancy, compressing two or more redundant components into a single one. This clustering is performed hierarchically to avoid evaluating all possible combinations (although they would ensure the optimal solution, the computational effort is not cost effective). In a simplified way, this clustering approach consists in clustering the entire model in two groups and approximating each group corresponding components by a single Gaussian. For each cluster group, we evaluate if the one Gaussian approximation deviates too much from the components it is representing. If the distance is too large, then this cluster is further clustered in two sub-groups. This evaluation proceeds iteratively until no sub-group one Gaussian approximation exceeds a predefined distance from it’s components.

Finally, we presented problems associated with the algorithm stability in high dimensions, and mitigation strategies for problematic non-normalized data. The next chapter will present the main contributions and decisions to handle these problems. Furthermore, we present extensions to the model to improve computational speed, reduce memory use, and enhance modeling power.
The online kernel density estimation approach presented before (Kristan, Leonardis, and Skočaj 2011) presents several desirable properties as it is non-parametric, does not require saving 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 changing and extending the algorithm very difficult and error prone. Furthermore, as the number of feature dimensions grow, performance heavily degrades, and memory use 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.

This chapter presents our version, xokde++, an efficient and numerically robust, object oriented online KDE implementation. It was implemented in C++ and uses a state of the art high-performance object-oriented algebra implementation (Guennebaud, Jacob, et al. 2010). This implementation includes, among others, matrix multiplication, singular value decomposition (SVD) and Eigenvalue Decomposition. Implementing xokde++ in an object-oriented programming (OOP) language allows for higher flexibility and abstraction advantages. In addition, C++ allows for efficient memory use by using move semantics, which effectively reduces the number of copies and memory allocations.

While we follow the theoretical model presented before, we made several choices 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.

### 4.1 Numeric Stability

Working with arbitrary data distribution and dimensionality, especially in long-running applications, is prone to several numeric issues that must be accounted for in order to allow the algorithm to recover from these situations. In this section we will approach several strategies that add numeric robustness to the algorithm, allowing it to work with skewed or degenerate sample distributions, and to handle and avoid underflows or overflows.
4.1.1 Determinant Computation

The Gaussian probability density function as defined in eq.3.2 uses the determinant ($|\Sigma|$) and the inverse ($\Sigma^{-1}$) of the covariance matrix. One way to efficiently compute the inverse of a non-singular matrix is to first compute its LU decomposition $A=LU$ and use it to compute the inverse.

This is also convenient, as we will need to compute the matrix’s 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 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 use double precision floating point numbers. To avoid this problem, 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

$$\log|\Sigma| = \sum_{i=1}^{D} \log(U_{ii})$$

where $D$ is the number of dimensions of the covariance matrix. Because we guarantee the covariance is positive definite, matrix $U$ also has size $D$.

4.1.2 Degenerate Covariance Matrices

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 covariance value of 0 along that axis. To correct this situation, we detect and correct singular or near singular matrices by computing its eigendecomposition

$$\Sigma = Q\Lambda Q^T, \ \Lambda_{ii} = \lambda_i$$

and checking for eigenvalues below a predefined threshold of $10^{-9}$. If any of the eigenvalues falls below, then a covariance revitalization algorithm is employed. This consists of 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. Considering the normalized version of $\Lambda$,

$$\hat{\Lambda}_{ii} = \hat{\lambda}_i = \left(\frac{\lambda_i}{\arg \max \lambda_i}\right)$$

$$\Lambda'_{ii} = \Lambda'_i = \begin{cases} \lambda_i & \text{if } \hat{\lambda}_i \geq 1E^{-9} \\ \alpha & \text{otherwise} \end{cases}$$

where $\alpha$ is 1% of the average of the non-problematic eigenvalues

$$\alpha = \frac{1}{10} \left(\left|\{\lambda_j : \hat{\lambda}_i > 1E^{-9}\}\right|\right)^{-1} \sum_{\hat{\lambda}_i > 10^{-9}} \lambda_i$$
4.1. NUMERIC STABILITY

The corrected covariance is then reconstructed using the corrected eigenvalues

\[ \Sigma' = Q\Lambda'Q^T \]  

(4.6)

4.1.3 Whitening Transformation

Estimating a single optimal bandwidth and using it in all components, means that each component is scaled equally in all directions. If the spread of data is much greater in one coordinate direction than the others, then it is probably best to pre-scale the data to avoid extreme differences of spread in the various coordinate directions. If this is done, then a single smoothing parameter will generally suffice. One approach, suggested by Fukunaga (1990), is to first pre-whiten the data by linearly transforming it to have unit covariance matrix, next to smooth using a radially symmetric kernel, and, finally, to transform it back. Figure 4.1 depicts the whitening approach. This approach allows the optimal bandwidth to better fit the global distribution of the data, leading to better models (Silverman 1986).

![Figure 4.1: Example of a non-white data (a), and its whitened version (b).](image)

Another advantage of the whitening transformation is in situations where some dimensions of the data are so highly correlated that they are measuring similar characteristics, and where we are not interested in those correlations per se. In this case, we want to sphere the system of coordinates as a whole, to create new coordinates that are linear combinations of the old and where all measure substantially different aspects of the data. This is useful in the compression phase which relies on a clustering approach and benefits from having a system of coordinates where each dimension is relevant and non-redundant.

With this in mind, and similarly to the oKDE implementation, both the optimal bandwidth and compression phases described in sections 3.2 and 3.3 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.

Formally, we can define the whitening transformation as

$$Y = Q^T X$$

where $Q$ are the eigenvectors of the covariance matrix $\Sigma$, such that its eigenvalues $\Lambda = I$. Sub-
stituting in eq. 4.2 results in

$$Q^T \Sigma Q = I$$

A more detailed explanation of the whitening transformation is available in Appendix B, but it
is important to note that, as long as $\Sigma$ is positive definite, and section 4.1.2 guarantees it is, $Q$
exists and is non-singular. Thus, the whitening transform is reversible.

The whitening transform impacts the model in two ways. The first is computational per-
formance, since the structure $F$ in equations 3.6, 3.12, and 3.15, will be the identity matrix:
thus, computational cost is significantly reduced as less matrix multiplications are needed. The
second advantage is model quality, as both the optimal bandwidth computation and the com-
pression phase, performed using the Goldberger K-means approach, benefit from working on
spherized data.

Since the whitening transformation is neutral, i.e., it can be performed forwards and back-
wards, we could apply the transformation to the model, 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, i.e., its underlying two Gaussian mixture. These selected com-
ponents are split into both main and secondary models. Then, in the Hierarchical Clustering
phase, the components will be grouped and marked for merging. 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.

As a final remark, we should note that, although not explicitly described in the published
work, the whitening transformation was present in the original oKDE implementation. Even
so, due to its potential importance to model quality and stability we felt relevant to include it
in the xokde++ description.

4.1.4 Optimal Bandwidth Stability

Sometimes, the observed data may produce zero-valued optimal bandwidths. Recall from
eq. 3.6 that we assume that the structure of the optimal bandwidth is well-approximated by
the covariance of the entire model and that finding the optimal bandwidth amounts to finding
the optimal scaling factor $\beta_{opt}$. However, there may be situations where this scaling factor is
zero. This may be caused by precision issues or by data distribution in the model. The point
is that when this scaling factor is zero, the produced optimal bandwidth is a zero matrix. This
is inconvenient as it means no smoothing will be applied to the KDE. On example are the
Dirac-deltas, that will remain with a zero-valued covariance matrix, not allowing the use of
the likelihood function. Recalling that the optimal bandwidth is computed on the whitened
data, one solution for this problem is to detect these occurrences and set the bandwidth as the identity matrix, and then computing the backwards whitening transformation.

4.2 Diagonal Covariance Matrices

Because the Gaussian components in the model are acting together to model the overall feature density, full covariance matrices may not be necessary, even if the features are not statistically independent. The linear combination of the components with diagonal covariance is capable of modeling the correlations between features. That is, the effect of using a set of $M$ Gaussians with full covariance can be obtained by using a larger set of diagonal covariance matrices.

In practical terms, using diagonal covariance matrices has two direct implications. First, it significantly reduces the amount of variables that need to be estimated. Recall that the number of variables grows quadratically with the number of dimensions if full matrices are used. This makes using diagonal covariance matrices a very effective solution for high dimensionality. Since the diagonal covariances ignore dependencies across dimensions, this could pose a problem of inadequate fitting the data. But, as stated before, this is a problem circumvented by using more diagonal components. It also reduces memory needs as only the vector corresponding to the diagonal matrix has to be stored. For some combinations of feature dimensionality and distribution, the memory gains may be less evident, as we are modeling non-linearities by compensating with more components. Still, in general, memory efficiency gains should be expected.

The second implication is also related to the assumption of independence across dimensions. For problematic non-normalized data, this might improve stability, as it ignores certain non-linear relationships that could result in singular covariance matrices. Furthermore, our numeric stability strategies have a stronger impact for diagonal covariances, as corrections are not diluted through the entire matrix, and are concentrated in the diagonal entries. For data with flat or near-flat dimensions this is particularly relevant, as will be shown later in chapter 6.

4.3 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. This means that the cost of adding a sample is simply that of adding one Dirac-delta 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, at each insertion the new computed bandwidth would replace the previous one, and, in the end, only the last computed bandwidth would actually be used for the compression phase.

The determinant and the inverse of the covariance matrix of a component is 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, making the process of adding a component to the mixture faster.

4.4 Code Extensibility and Integration

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 an 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 triangular covariance matrices, by simply writing a new class and instantiating the template accordingly. The same logic applies if we want to replace the Gaussian pdf by a different one. Instead of Gaussians, we could use other probability density functions, such as the Student-t, by implementing the corresponding class and respecting the current interfaces. Similarly, changing the compression phase, currently performed using a clustering approach, by another, is also possible, as long as the interfaces are respected.

To finalize, we note that our implementation is a pure template library and that there are no dependencies other than C++ standard library and Eigen 3.x, which is also a template library. Our implementation can thus be integrated as a component of some larger project simply by including the necessary xokde++ header files.

4.5 Summary

In the previous chapter, we presented the base algorithm that was implemented in xokde++. This chapter presented the strategies adopted to improve numerical stability and computational speed along with discussions of why they are needed and how they affect model estimation.

Regarding numeric stability, four main aspects were studied. First, determinant computation, in which we directly compute the log determinant and use it to compute the log likelihood function instead of the traditional likelihood function. Second, degenerate covariance matrices are corrected to be positive definite. Third, the whitening transformation is described as it helps both stability and model quality, due to some of the phases in the algorithm benefiting from working on spherized data. Finally, we present a recovery approach for the optimal bandwidth when the scaling factor goes to zero.

To improve computational speed, two relevant approaches were described. The lazy operation, where each computation is only performed when needed, and result buffering, where we save results, such as the covariance inverse, for later reuse.

In this chapter we also described one variation of the original oKDE algorithm which is using diagonal covariance matrices. This approach is able to improve both numeric stability and computational speed, as the number of variables to estimate are much lower.

Finally, we presented considerations about code extensibility, exemplified by the diagonal covariance variation, and ability to be integrated in larger projects, as xokde++ is a pure
template library.

Chapter 5 will present the evaluation setup by describing the metrics, comparative approaches, and datasets used in the evaluation.
Evaluation Setup

The original oKDE approach (Kristan, Leonardis, and Skočaj 2011) is a good approach for online kernel density estimation. However, its implementation suffers from numerical and computational performance issues. Our work provides an efficient and robust implementation of the original oKDE approach. In the end, we expect to produce models with similar quality while taking a fraction of the time and memory needed by the original implementation. Even though the original description is closely followed, there are implementation details that may be different from those of the original authors, and thus, the final models may be different. Plus, some algebra routines implemented in Eigen 3.x are not the same as the ones implemented in MATLAB. One such example lies with the singular value decomposition (SVD) computation, in which the Eigen’s implementation produces an equally valid, but different decomposition. These aspects directly impact the final model produced by the training algorithms and prevent us from producing exactly equal models.

Nevertheless, note that this is not an intrinsic problem. To properly evaluate our implementation and extensions, as well as a way of maximizing the comparability of the results, we decided to follow the main strategies of the original oKDE paper (Kristan, Leonardis, and Skočaj 2011). 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 by employing the estimated models in classifier based tasks as in the original work.

In addition to the oKDE and xoke++, which are both generative methods, we decided to compare the classification results with a discriminative approach, which are typically better suited for classification tasks. Again with the reproducibility of the original oKDE work in mind, we used the online SVM (OISVM) (Orabona, Castellini, Caputo, Jie, and Sandini 2010) as implemented in the DOGMA (Orabona 2009) MATLAB toolbox, as this was one of the discriminative online approaches the original oKDE authors used.

For all evaluations, we chose varied datasets. These correspond not only to the datasets used in the original papers (for comparative evaluation), but also a new dataset for assessing the quality of high dimensional models. Some datasets have been updated in recent years. Thus, in order to have comparable results between our implementation and the original one, we ran both approaches in the same, most recent versions of the original datasets. In addition, we also evaluate performance on The L2f Face Database, specifically collected to allow us to test the high dimensional scenario, and that neither the original oKDE implementation nor OISVM could handle.

Since we want to perform comparative evaluation, we reproduce the results obtained in the original by oKDE, by following the same evaluation setup. Thus, for the online classifier training task, we randomly shuffled the data in each dataset and used 75% of the data to train and the remaining to test. For each dataset, we generated 12 such random shuffles to minimize
the impact of lucky/unlucky partitions. To maximize comparability, the same 12 shuffles given to the MATLAB oKDE implementation are given to xokde++ and OISVM.

For the kernel density estimation approaches (oKDE and xokde++), each class is represented by the KDE model resulting from using its 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. At the end of the training phase we end up with \( k \) models, corresponding to \( k \) classes. To evaluate classification performance we use a simple Bayesian criterion

\[
\hat{y} = \arg \max_k p(x|c_k)p(c_k)
\]

(5.1)

where \( c_k \) corresponds to the kde trained with the samples from the \( k \)'th class, \( p(x|c_k) \) 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 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 multiclass approach was followed

\[
\hat{y} = \arg \max_k f_k(x)
\]

(5.2)

in which \( f_k(x) \) provides the confidence score of the \( k \)'th binary classifier for sample \( x \). We used a second order (quadratic) polynomial kernel with \textit{gamma} and \textit{coefficient} 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, where unknown classes and data are bound to appear. In a sense, a training set does not even exist a priori, and the classifier is to be trained continuously as more samples arrive. Preliminary experiments also showed that for several datasets, changing the \( C \) parameter did not significantly change the result, 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 previously stated, we wish to make minimal assumptions about the nature of data or about its 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 exclusive responsibility of the internals of each approach.

5.1 UCI Datasets

The authors of the original oKDE papers (Kristan, Leonardis, and Skočaj 2011; Kristan and Leonardis 2014) 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 (Lichman 2013) and differ in length, data dimensionality, and number of classes. Table 5.1 presents a
summarized view of each dataset’s properties and Fig. 5.1 shows the balance of classes in each dataset.

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

Table 5.1: 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)

5.2  High Dimensional Task

Since we aim at continuous modeling and adaptation of incoming information, we want to be able to work both low and high-dimensional scenarios with minimal restrictions regarding feature nature or distribution. Face recognition tasks provide a good testing ground for the high-dimensional scenario, as they typically are high dimensional problems and situations such as face pose and expression provide wide variations in feature distribution. With this in mind, we survey the face recognition task regarding feature choice and strategies to mitigate high dimensionality. We stress that facial recognition is not our main task: our ambition is to address the general high-dimensional data modeling problem. Other approaches, specifically tailored for face recognition, will probably be more effective for some facial recognition task, but inapplicable for generic unsupervised applications.

Traditional face recognition systems are focused on solving problems of automatically classifying human faces from multiple poses, varying expression, and illumination, while at the same time, using a relatively low amount of training information per subject. Basically, what they do is measure the distance from the test images to the model image (Turk and Pentland 1991) or images (Lee and Kriegman 2005) of each subject. The models are usually trained offline in a supervised way, using all available possible information. Updates to the model typically require a full retrain. Most systems assume the data is already cropped and aligned (Turk and Pentland 1991; Wright, Yang, Ganesh, Sastry, and Ma 2009) while others automatically crop and align the images (Mian 2011). For evaluation purposes, the vast majority of these algorithms rely on picture databases, such as Color FERET (Phillips, Wechsler, Huang, and Rauss 1998), Yale-B (Georghiades, Belhumeur, and Kriegman 2001), extended Yale-B (Lee, Ho, and Kriegman 2005), AR (Martinez 1998), or ORL (Samaria and Harter 1994), to name a few.
Mian (2011) provides an example of an attempt to use video streams, but they address the problem of memory limitation from continuously obtaining image samples by discarding old information. To our knowledge, there are no attempts to continuously model faces coming from video streams without discarding old information and continuously integrating all data samples. Discarding old information may be undesirable in situations where we are modeling data characteristic that may not reappear for very long times. One example from facial recognition would be hair, or beard, which might be long for a long time, and then cut, and only much later will be long again. If we discard the information regarding a long beard, we might lose some ability to accurately represent that subject at all times.

Our main goal aims to continuously model faces without discarding old information, in an unsupervised way, and still be able to build a discriminative system on top of this model. However, to maximize the discriminative power of the underlying model, we need to choose appropriate features. These should be highly discriminative, robust to noise, have low computational complexity and, if possible, be compact, i.e., low to medium number of dimensions. This later requirement, although optional, is desirable, as we have already described the problems that high dimensionality brings to estimation algorithms for GMMs.

The simplest and most direct way to obtain image features is to directly use the pixel’s color or gradient if the image is in grayscale. This is, by far, the most used feature as it is both simple and fast to get. The only preprocessing needed is to rescale the image, using, for example, bilinear interpolation, to reduce the number of pixels we are working with.

Below, we present some of the most relevant methods, as well as their features.
5.2. HIGH DIMENSIONAL TASK

Eigenfaces and Fisherfaces share the idea that we can display the pixels column-wise and have each image be one column of a matrix. Then, applying standard algebra matrix transformations, we can obtain a discriminative low-dimensional feature vector. Eigenfaces come from Principal Component Analysis (PCA), in which the eigenvectors with the highest associated eigenvalue are selected (eigenfaces) (Turk and Pentland 1991). Similarly, Linear Discriminant Analysis (LDA) is used to compute fisherfaces (Bellhumeur, Hespanha, and Kriegman 1997), named in honor of the LDA author, Ronald Fisher. The way these feature vectors are obtained typically requires the availability of the full training set, which makes these kind of approaches not eligible for our needs.

Local Binary Patterns (LBP) (Ojala, Pietikäinen, and Harwood 1996; Ahonen, Hadid, and Pietikäinen 2004; Ahonen, Hadid, and Pietikainen 2006) were introduced as a means of summarizing local structure in gray level images, and have proved to be highly discriminative for texture classification. The LBP idea is to take a pixel neighborhood and threshold at the value of the central pixel and, then, use the resulting binary valued image patch as a local image description, figure 5.2. Formally we have,

$$LBP(x_c, y_c) = \sum_{p=0}^{P-1} 2^p s(g_p - g_c),$$

where $P$ is the number of pixels in the neighborhood (8 in a typical $3 \times 3$ neighborhood, giving an 8-bit pattern), $g_p$ and $g_c$ are the gray-level values and $s(x)$ is defined by eq. 5.4.

$$s(x) = \begin{cases} 1 & \text{if } x \geq 0; \\ 0 & \text{otherwise} \end{cases}$$

Neighborhoods of different sizes allow dealing with textures at different scales. Ojala, Pietikainen, and Maenpaa (2002), also defined the so called uniform patterns, in which an LBP is considered uniform if it contains at most one 0-1 and one 1-0 transition when viewed as a circular bit string. The uniformity is important because it characterizes the patches containing primitive information such as corners and edges. Although only 58 of the 256 8-bit patterns are uniform, the authors concluded that nearly 90% of the image neighborhoods are uniform, and the remaining are essentially noise. Thus, when calculating the histograms for the LBPs, all non-uniform patterns can be put in the same bin without losing much information. These histograms will then serve as the image descriptors, and will be compared to look for similarity.

Tan and Triggs (2010) proposed Local Ternary Patterns (LTP) as an improvement of LBPs. The goal was to make them less sensitive to noise. For this, instead of thresholding exactly on the central pixel value, they set a tolerance interval $[x - t, x + t]$ around the central pixel value. Furthermore, $s(x)$ is replaced as described by eq. 5.5.
CHAPTER 5. EVALUATION SETUP

Figure 5.3: Illustration of the basic LTP operator.

\[
s(x, i_c, t) = \begin{cases} 
1 & x \geq i_c + t; \\
0 & |x - i_c| < t; \\
-1 & x \leq i_c - t 
\end{cases} 
\] (5.5)

where \(i_c\) is the central point, \(x\) is the neighbor and \(t\) is the tolerance parameter. Fig. 5.3 provides a graphic example of this process. As with the LBP\(s\), the uniform pattern argument is valid, so the authors split each ternary pattern into its positive and negative halves (figure 5.4). They then have two separate channels, each representing the original image patch, that can be treated separately.

Figure 5.4: Illustration of the two channel LTP operator.

Lowe (2004) proposed the scale-invariant feature transform (SIFT) with the goal to detect keypoints that are robust against image translation, rotation, and scaling. Mian (2011) presents an example of an application of these features to a face recognition task. The keypoints are detected at the scale space extrema in the difference-of-Gaussian function convolved with the image. They must also satisfy other conditions, including high contrast, and good localization along an edge. Note that the detected keypoints do not correspond to any specific face landmark, e.g. nose or mouth. For each keypoint, regions of \(4 \times 4\) pixels are used to create orientation histograms with eight bins, forming a 128-dimension vector. An example of a \(2 \times 2\) keypoint descriptor is given in figure 5.5.

Dominant gradient directions are used, making it rotation invariant. For illumination robustness, the vector is normalized to unity, thresholded at a 0.2 ceiling and re-normalized to unit length.

However, as noted by Liao, Jain, and Li (2013), SIFT keypoints may not be ideal for face recognition as only a limited number of keypoints are detected for face images. Due to the similarity of face images, this lower number of keypoints may not be sufficient to effectively discriminate between the two faces.

(Liao, Jain, and Li 2013), propose Gabor Ternary Patterns (GTP) as a more suitable way to represent face features. Gabor filters were chosen due to their ability to provide a good
5.2. HIGH DIMENSIONAL TASK

Figure 5.5: To create a keypoint descriptor we first compute the gradient magnitude and orientation at each image sample point around the keypoint, as shown on the left. These gradients are then weighted by a Gaussian window, indicated by the overlaid circle. Finally, these samples are accumulated into orientation histograms, summarizing the contents over $4 \times 4$ sub-regions, as shown on the right. The length of the arrows corresponds to the sum of the gradient magnitudes near that direction within the region. For graphical simplicity, the picture depicts a $2 \times 2$ descriptor computed from an $8 \times 8$ set of samples.

Figure 5.6: Comparison of SIFT and CanAff keypoint detectors. Left: 37 keypoints detected by SIFT. Right: 571 keypoints detected by CanAff (only the first 150 are depicted).

First, keypoints are detected using CanAff, the authors edge-based detector, which extracts many more keypoints than the SIFT detector (figure 5.6), and constructs $40 \times 40$ regions using the neighborhoods. After normalizing the regions, they construct a local descriptor within each detected region. To give a very simplified description, this descriptor is built by first applying a Gabor filter (Wiskott, Fellous, Kuiger, and Von Der Malsburg 1997; Liu and Wechsler 2002) with four orientations to the region. The Gabor kernels are defined in eq. 5.6.

$$
\psi_{\mu, \nu}(x, y) = \frac{||k_{\mu, \nu}||}{\sigma^2} \exp \left(-\frac{||k_{\mu, \nu}||^2||z||^2}{2\sigma^2}\right) \times \left[ \exp(i, k_{\mu, \nu}^\top z) - \exp\left(-\frac{\sigma^2}{2}\right) \right],
$$

(5.6)

where $\mu$ and $\nu$ define the orientation and scale of the Gabor kernel, $z = (x, y)^\top$, and the wave vector $k_{\mu, \nu}$ is defined in eq. 5.7.

$$
k_{\mu, \nu} = (k_{\nu} \cos \phi_{\mu}, k_{\nu} \sin \phi_{\mu})^\top,
$$

(5.7)
with $k_{\nu} = (\pi/2)/(\sqrt{2})^{\nu}$ and $\phi_\mu = \pi\mu/8$ (Wiskott, Fellous, Kuiger, and Von Der Malsburg 1997; Liu and Wechsler 2002). Due to the relatively small region size, the Gabor kernels are processed using a single scale ($\nu = 0$) and four orientations ($\mu \in 0, 2, 4, 6$), corresponding to 0, 45, 90, and 135 degrees, with $\sigma = 1$. Furthermore, the authors use odd Gabor kernels (imaginary part) which is sensitive to edges and their locations. For each pixel, there are four Gabor filter responses, described in eq. 5.8,

$$f_i(x, y) = G_i(x, y) \ast I(x, y), \quad i = 0, 1, 2, 3,$$

where $G_i(x, y)$ is the $i$th odd Gabor kernel and $\ast$ is the convolution operator. The responses of the four filters are then combined as a ternary pattern (Tan and Triggs 2010) as described in eq. 5.9.

$$GTP_t(x, y) = \sum_{i=0}^{3} 3^i \left[ f_i(x, y) < -t \right] + 2 \left[ f_i(x, y) > t \right],$$

where $t$ is a small positive threshold, 0.03 in the authors’ paper. This local descriptor, named GTP, encodes local structures from the responses of the Gabor filters in four different orientations, and the local ternary pattern provides a discriminative encoding of the four Gabor features responses. There are a total of $3^4 = 81$ possible GTP patterns. Then, the $40 \times 40$ region is divided into $4 \times 4 = 16$ sub-grid cells, each with a size of $10 \times 10$ pixels. A histogram of GTPs is calculated for each sub-grid cell, and concatenated to form a 1,296-D ($4 \times 4 \times 81$) feature vector. After normalization, to reduce the influence of outliers, PCA is applied, reducing the feature dimension to 128-D. The GTP approach has proved to perform better than SIFIs for face recognition, particularly in uncontrolled scenarios.

5.2.1 Final considerations

Many more features for face recognition exist in the literature, and an exhaustive description is beyond the scope of this thesis. However, we would like to emphasize that, among this abundance of approaches, there are features that are more descriptive and tuned for each specific task. For other tasks, other features would be more adequate. Additionally, some features, although more descriptive, increase the computational complexity, which can make them unsuitable for an online system. Approaches exist that rely on statistical models (Cox and Pinto 2011) to select the best feature set. Although potentially finding more descriptive features, these models are, in principle, not easily suited for our setup, due to computational complexity.
issues. This is not to say they cannot be used: using these techniques to find the best set of features for a given task, and incorporating that knowledge into the online system, by selecting or prioritizing those features \textit{a priori} is one way of directly leveraging these kind of approaches.

Another insight we gained from face recognition literature is that sparse representations (Wright, Yang, Ganesh, Sastry, and Ma 2009; Yang, Zhang, Yang, and Zhang 2010; Yang, Zhang, and Yang 2011; Zhang, Yang, and Feng 2011; Wright, Yang, Ganesh, Sastry, and Ma 2009; Gijsberts and Metta 2013) with dictionaries (Yang, Lv, Ren, Yang, and Jiao 2014; Mairal, Bach, Ponce, and Sapiro 2009; Zhang and Li 2010; Yang, Zhang, Shiu, and Zhang 2013) can be a powerful way to increase the discriminative power of a classifier. In addition, if we use the GMM to extract the \textit{a posteriori} probability vector, we obtain a vector composed mostly by zero or close to zero values, since the highest probabilities go to the nearest Gaussians. This vector can be viewed as a sparse representation of the observation, and used to build the dictionary. Although sparseness and dictionaries will not be addressed, we will keep this in mind for potential opportunities for improvement of the recognition task. Note that the dictionaries can be trained online (Yang, Lv, Ren, Yang, and Jiao 2014; Mairal, Bach, Ponce, and Sapiro 2009), which makes researching the possibility of integration with our online KDE a long-term option. Yang, Yu, and Huang (2010) provides some clues on how to achieve this.

5.3 \textbf{L2F Face Database}

The L2F 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 (Bradski 2000) face detector, which uses a Haar Cascade (Viola and Jones 2001) approach for face detection and cropping. 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 5.8 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 restrictions 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 do not even need to be saved to 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 5.9 presents an example of the impact of the various crop operations in a sample image.

Figure 5.8: Some examples of the dataset pose and expression variability.

Figure 5.9: All crop schemes, 0, 2, 4, 8, 16, and the masking approach to extract 128 informative pixels.

5.4 Summary

This chapter presented the datasets used in the evaluation procedure, including a new dataset specifically captured in the context of this work to test the performance in high dimensional space. The evaluation setup was also described, consisting on three major aspects. The first, intrinsic evaluation, is concerned with model complexity and sample prediction likelihood. The second, extrinsic evaluation, is concerned with classification accuracy in a classification task. Finally, computational performance, comprises analyses of both time and memory performance. These metrics allow us to comprehensively compare our approach with the state of the art oKDE approach.

In the next chapter, results and conclusions for each of these metrics will be presented,
as well as a detailed comparison between the full and the diagonal covariance variants of xokde++.
Results and Discussion

In this chapter, the performance of the original oKDE, xokde++, and OISVM are evaluated under various perspectives. In order to compare our performance to the original oKDE work, much of the evaluation setup, described in the previous chapter, closely follows the original evaluation. As such, we measure the model quality, both extrinsically and intrinsically, and the computational performance, including time and memory requirements. We compare oKDE with our own implementation xokde++ with both full and diagonal (xokde++/diag) covariances. This allows us to observe what is the penalty, if any, on model quality of forcing diagonal covariances to improve computational speed. Finally, we apply and evaluate xokde++ in the high dimensional task of face recognition.

6.1 Experimental Setup

All experiments were performed on an Intel Xeon CPU E5530 @ 2.40GHz with 24GB RAM, running under Linux OpenSuse 13.1 operating system. Both oKDE and OISVM were run using MATLAB version R2013a using options: -nosplash, -nodesktop, -nodisplay, and -nojvm. Our xokde++ was compiled using c++1y and enabling all vectorizing options available to the our CPU architecture.

6.2 Model Quality

To measure estimation quality, we use the average negative log-likelihood. Which is computed by using the negative log-likelihood of the KDE model corresponding to the true label of each of the $N$ samples.

$$-\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} -\log(p_{\text{KDE}}(x_i))$$  \hspace{1cm} (6.1)

We also compare of the average complexity of the models. Table 6.1 and figure 6.1 present 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 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 recovering from these situations by correcting the covariances. This is done by inducing a small variance, computed using currently available information, until more data arrives allowing for a more accurate estimation.
### Table 6.1: 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.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>xokde++</th>
<th>xokde++/diag</th>
<th>oKDE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Complexity</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iris</td>
<td>28 ± 3</td>
<td>22 ± 3</td>
<td>28 ± 3</td>
</tr>
<tr>
<td>Yeast</td>
<td>31 ± 15</td>
<td>30 ± 19</td>
<td>31 ± 15</td>
</tr>
<tr>
<td>Pima</td>
<td>62 ± 9</td>
<td>42 ± 19</td>
<td>64 ± 7</td>
</tr>
<tr>
<td>Winequality-red</td>
<td>39 ± 23</td>
<td>53 ± 37</td>
<td>40 ± 24</td>
</tr>
<tr>
<td>Winequality-white</td>
<td>31 ± 24</td>
<td>54 ± 40</td>
<td>36 ± 25</td>
</tr>
<tr>
<td>Wine</td>
<td>44 ± 7</td>
<td>44 ± 7</td>
<td>45 ± 7</td>
</tr>
<tr>
<td>Letter</td>
<td>65 ± 12</td>
<td>42 ± 9</td>
<td>66 ± 14</td>
</tr>
<tr>
<td>Image Segmentation (seg)</td>
<td>49 ± 12</td>
<td>51 ± 24</td>
<td>52 ± 11</td>
</tr>
<tr>
<td>Steel</td>
<td>8 ± 2</td>
<td>20 ± 7</td>
<td>4 ± 1</td>
</tr>
<tr>
<td>Breast Cancer</td>
<td>40 ± 7</td>
<td>153 ± 11</td>
<td>50 ± 12</td>
</tr>
<tr>
<td>Skin</td>
<td>8 ± 2</td>
<td>10 ± 2</td>
<td>8 ± 2</td>
</tr>
<tr>
<td>Covtype</td>
<td>18 ± 5</td>
<td>17 ± 5</td>
<td>20 ± 5</td>
</tr>
<tr>
<td><strong>-\mathcal{L}</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iris</td>
<td>7.4 ± 1.7</td>
<td>3.3 ± 1.3</td>
<td>7.8 ± 1.6</td>
</tr>
<tr>
<td>Yeast</td>
<td>-13.2 ± 0.5</td>
<td>-14.3 ± 0.3</td>
<td>-7.1 ± 1.7</td>
</tr>
<tr>
<td>Pima</td>
<td>29.3 ± 0.5</td>
<td>27.6 ± 0.2</td>
<td>31.0 ± 0.7</td>
</tr>
<tr>
<td>Winequality-red</td>
<td>-0.2 ± 0.6</td>
<td>0.2 ± 0.3</td>
<td>14.7 ± 4.2</td>
</tr>
<tr>
<td>Winequality-white</td>
<td>0.5 ± 0.2</td>
<td>1.6 ± 0.2</td>
<td>55.2 ± 54.3</td>
</tr>
<tr>
<td>Wine</td>
<td>51.1 ± 4.3</td>
<td>33.0 ± 3.4</td>
<td>64.9 ± 18.5</td>
</tr>
<tr>
<td>Letter</td>
<td>17.9 ± 0.1</td>
<td>19.9 ± 0.1</td>
<td>18.2 ± 0.1</td>
</tr>
<tr>
<td>Image Segmentation (seg)</td>
<td>25.7 ± 0.8</td>
<td>31.0 ± 1.0</td>
<td>156.6 ± 33.7</td>
</tr>
<tr>
<td>Steel</td>
<td>–</td>
<td>85.2 ± 4.3</td>
<td>–</td>
</tr>
<tr>
<td>Breast Cancer</td>
<td>-25.6 ± 5.7</td>
<td>-16.4 ± 2.4</td>
<td>433.9 ± 33.7</td>
</tr>
<tr>
<td>Skin</td>
<td>13.5 ± 0.3</td>
<td>13.9 ± 0.2</td>
<td>13.4 ± 0.2</td>
</tr>
<tr>
<td>Covtype</td>
<td>51.9 ± 0.1</td>
<td>55.9 ± 0.0</td>
<td>52.6 ± 0.1</td>
</tr>
</tbody>
</table>
Figure 6.1: Average negative log-likelihood given by the model corresponding to the true label. Note that to improve readability there is a break in the series for the highest log-likelihood measurement.
Figure 6.2: Average model complexity, along with the standard deviation over the 12 fold set.
6.3. CLASSIFIER ACCURACY

To analyse the discriminative properties of the implementations, we constructed classifiers as described in chapter 5. As shown in table 6.3, 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 for 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, since this approach 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 when handling non-normalized datasets.

Results for the online independent support vector machine (OISVM) are also provided as it is useful 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 much. The table also shows that OISVM has similar numeric instabilities as the oKDE implementation, failing on the same datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>xokde++</th>
<th>xokde++/diag</th>
<th>oKDE</th>
<th>OISVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>96.4 ± 2.7</td>
<td>95.0 ± 3.4</td>
<td>96.4 ± 2.4</td>
<td>97.1 ± 2.1</td>
</tr>
<tr>
<td>Yeast</td>
<td>49.7 ± 2.3</td>
<td>48.1 ± 1.6</td>
<td>50.6 ± 3.3</td>
<td>59.2 ± 2.2</td>
</tr>
<tr>
<td>Pima</td>
<td>67.8 ± 3.4</td>
<td>70.1 ± 3.9</td>
<td>69.7 ± 2.9</td>
<td>76.9 ± 2.5</td>
</tr>
<tr>
<td>Winequal-red</td>
<td>62.0 ± 2.5</td>
<td>54.6 ± 1.9</td>
<td>56.9 ± 6.3</td>
<td>58.3 ± 1.9</td>
</tr>
<tr>
<td>Winequal-white</td>
<td>49.9 ± 1.3</td>
<td>42.4 ± 1.7</td>
<td>44.9 ± 10.6</td>
<td>53.2 ± 1.6</td>
</tr>
<tr>
<td>Wine</td>
<td>97.7 ± 1.4</td>
<td>98.5 ± 1.8</td>
<td>93.9 ± 6.1</td>
<td>96.8 ± 2.8</td>
</tr>
<tr>
<td>Letter</td>
<td>95.8 ± 0.2</td>
<td>93.4 ± 0.4</td>
<td>95.8 ± 0.2</td>
<td>93.0 ± 0.4</td>
</tr>
<tr>
<td>Image Seg.</td>
<td>91.5 ± 1.1</td>
<td>89.4 ± 1.2</td>
<td>75.0 ± 5.3</td>
<td>95.0 ± 0.9</td>
</tr>
<tr>
<td>Steel</td>
<td>31.9 ± 11.0</td>
<td>56.9 ± 9.0</td>
<td>8.7 ± 1.0</td>
<td>24.3 ± 12.1</td>
</tr>
<tr>
<td>Breast Cancer</td>
<td>94.8 ± 1.7</td>
<td>96.2 ± 1.7</td>
<td>52.8 ± 12.0</td>
<td>95.9 ± 0.8</td>
</tr>
<tr>
<td>Skin</td>
<td>99.6 ± 0.1</td>
<td>99.4 ± 0.0</td>
<td>99.7 ± 0.1</td>
<td>99.8 ± 0.0</td>
</tr>
<tr>
<td>Covtype</td>
<td>52.0 ± 1.2</td>
<td>51.6 ± 0.6</td>
<td>68.0 ± 0.9</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 6.2: Results for average classification accuracy along with ± one standard deviation.

6.3 Classifier Accuracy
Figure 6.3: Average accuracy on each fold. The low results on the steel dataset are explained by the numeric instabilities of the algorithm. The missing column in the covtype dataset is because OISVM training has not finished.
6.4 Time and Memory Performance

The previous results show that our implementation produces models with similar or better quality than the original one. While these are good results, it is important to also note that our implementation does so at lower memory cost and much faster computation time. Table 6.3 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 greater, 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 use, when compared to oKDE, we use at most 10% of the memory required by the original implementation. This difference is even more noticeable in large datasets such as the skin and covtype. For these datasets, the oKDE implementation needed 913MB and 5064MB, respectively, while ours needed just 83MB and 361MB.

The OISVM approach has a tendency to need more memory as the number of classes increases. The reason for this is that since the OISVM trains a binary classifier, to adapt for multiclass problems we need to train \( K \) classifiers. The letter dataset is an example of that, since it has 26 classes, the memory needed by OISVM relative to oKDE was 4 times larger. The contrast is even higher compared to xokde++, where OISVM requires from 43 to 79 times more memory than the full or diagonal covariance approaches, respectively. For large datasets, time performance also degrades: the skin dataset took nearly twice the time it took for oKDE. This degradation of computational performance in OISVM gets prohibitive for the covtype dataset, which has 500,000 samples, and where 30 days of computation were not enough to complete training a single shuffle, let alone the 12 shuffles from our evaluation setup.

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

Table 6.3: Results for time performance along with ± one standard deviation. Time is presented in seconds. Speedups over the baseline oKDE approach are shown in column \( \rho \).
<table>
<thead>
<tr>
<th>Dataset</th>
<th>xokde++ $\rho$</th>
<th>xokde++/diag $\rho$</th>
<th>oKDE $\rho$</th>
<th>OISVM $\rho$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>4.6 3.76%</td>
<td>4.5 3.68%</td>
<td>123.0</td>
<td>116.6</td>
</tr>
<tr>
<td>Yeast</td>
<td>6.8 3.93%</td>
<td>5.8 3.35%</td>
<td>173.1</td>
<td>148.9</td>
</tr>
<tr>
<td>Pima</td>
<td>6.2 3.94%</td>
<td>5.2 3.29%</td>
<td>157.5</td>
<td>124.8</td>
</tr>
<tr>
<td>Winequality-red</td>
<td>8.1 5.60%</td>
<td>6.2 4.24%</td>
<td>145.4</td>
<td>162.2</td>
</tr>
<tr>
<td>Winequality-white</td>
<td>10.3 5.05%</td>
<td>8.3 4.08%</td>
<td>204.3</td>
<td>228.4</td>
</tr>
<tr>
<td>Wine</td>
<td>6.3 4.38%</td>
<td>4.8 3.32%</td>
<td>144.3</td>
<td>120.2</td>
</tr>
<tr>
<td>Letter</td>
<td>42.6 9.83%</td>
<td>23.2 5.36%</td>
<td>433.3</td>
<td>1823.9</td>
</tr>
<tr>
<td>Image Segmentation (seg)</td>
<td>15.7 7.72%</td>
<td>7.7 3.75%</td>
<td>203.9</td>
<td>192.3</td>
</tr>
<tr>
<td>Steel</td>
<td>10.5 5.77%</td>
<td>7.2 3.97%</td>
<td>181.3</td>
<td>194.5</td>
</tr>
<tr>
<td>Breast Cancer</td>
<td>10.6 5.50%</td>
<td>6.6 3.40%</td>
<td>193.1</td>
<td>141.8</td>
</tr>
<tr>
<td>Skin</td>
<td>83.1 9.10%</td>
<td>83.2 9.11%</td>
<td>913.3</td>
<td>2255.1</td>
</tr>
<tr>
<td>Covtype</td>
<td>361.2 7.13%</td>
<td>360.4 7.12%</td>
<td>5064.8</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 6.4: Results for memory performance along with relative usage compared to the baseline oKDE approach, shown in column $\rho$. Memory is presented in Megabytes.
Figure 6.4: Boxplots for accuracy on small to medium feature dimension (4 to 13 dims) datasets
Figure 6.5: Boxplots for accuracy on large feature dimension (16 to 30 dims) datasets. Skin and Covtype, although having smaller dimension (3 and 10) are also presented as they have a very large number of samples.
Figure 6.6: Average time required to train and test a single fold. Time is measured in seconds and presented in a logarithmic scale. Results for the steel dataset should be read with care, as OISVM algorithms did not converge due to numeric issues, failing early. The missing column in the covtype dataset is due to the OISVM not converging in acceptable time.
Figure 6.7: Average memory requirements for each fold. Memory is measured in megabytes and presented in a logarithmic scale. The missing column in the covtype dataset is due to the OISVM not converging in acceptable time.
6.5 Performance of Full vs Diagonal Covariances

The previous results show that our implementation produces consistently equivalent or better models than the original oKDE implementation at lower computational and lower memory cost. Moreover, our implementation also allows to choose between full and diagonal covariances. Table 6.5 presents a detailed comparison of the impact of using diagonal covariances regarding model quality and computational requirements. The table shows that using diagonal covariances produces models with slightly higher complexity, while exhibiting only a 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 affects the memory requirements, since the number of variables increases quadratically with the number of dimensions. Thus, datasets with higher dimensionality will have greater benefits from using diagonal covariances. However, since the diagonal approach tends to need more components, this, may partially offset the previous gain. Even so, the median value for memory use needed by the diagonal approach is 78% of the needed for full covariance. The gain 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 (ignoring the 25% improvement of the diagonal approach for the steel dataset). Considering all datasets, except the steel dataset, which is indicative for the diagonal approach numeric robustness but not useful 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.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>xokde++</th>
<th>xokde++/diag</th>
<th>acc</th>
<th>K</th>
<th>-L</th>
<th>time</th>
<th>mem</th>
<th>acc</th>
<th>K</th>
<th>-L</th>
<th>time</th>
<th>memory</th>
<th>AAccD</th>
<th>K diff</th>
<th>TS</th>
<th>mem use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>96.4%</td>
<td>95.0%</td>
<td>22</td>
<td>3.3</td>
<td>0.1</td>
<td>4.5</td>
<td>-1.4%</td>
<td>-7</td>
<td>3.1</td>
<td>97.9%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yeast</td>
<td>49.7%</td>
<td>48.1%</td>
<td>30</td>
<td>-14.3</td>
<td>6.5</td>
<td>5.8</td>
<td>-1.6%</td>
<td>-1</td>
<td>6.0</td>
<td>85.2%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pima</td>
<td>67.8%</td>
<td>70.1%</td>
<td>22</td>
<td>37.6</td>
<td>3.3</td>
<td>5.2</td>
<td>2.3%</td>
<td>-21</td>
<td>5.1</td>
<td>83.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Winequal-red</td>
<td>62.0%</td>
<td>54.6%</td>
<td>53</td>
<td>14.4</td>
<td>6.2</td>
<td>-7.4%</td>
<td>14</td>
<td>4.2</td>
<td>75.8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Winequal-white</td>
<td>49.9%</td>
<td>42.4%</td>
<td>54</td>
<td>39.0</td>
<td>6.5</td>
<td>5.8</td>
<td>-1.6%</td>
<td>-1</td>
<td>6.0</td>
<td>85.2%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wine</td>
<td>97.7%</td>
<td>98.5%</td>
<td>44</td>
<td>33.0</td>
<td>0.5</td>
<td>4.8</td>
<td>0.8%</td>
<td>0</td>
<td>6.6</td>
<td>75.8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Letter</td>
<td>95.8%</td>
<td>93.4%</td>
<td>42</td>
<td>19.9</td>
<td>191.7</td>
<td>23.2</td>
<td>-2.4%</td>
<td>-23</td>
<td>11.1</td>
<td>54.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image Seg.</td>
<td>91.5%</td>
<td>89.4%</td>
<td>51</td>
<td>31.0</td>
<td>45.0</td>
<td>7.7</td>
<td>-2.2%</td>
<td>-2</td>
<td>6.6</td>
<td>48.7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Steel</td>
<td>31.9%</td>
<td>56.9%</td>
<td>20</td>
<td>85.2</td>
<td>8.7</td>
<td>7.2</td>
<td>24.9%</td>
<td>12</td>
<td>22.0</td>
<td>68.8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Breast Cancer</td>
<td>94.8%</td>
<td>96.2%</td>
<td>153</td>
<td>-16.4</td>
<td>11.7</td>
<td>6.6</td>
<td>1.5%</td>
<td>113</td>
<td>4.5</td>
<td>61.8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skin</td>
<td>99.6%</td>
<td>99.4%</td>
<td>10</td>
<td>13.9</td>
<td>192.2</td>
<td>83.2</td>
<td>-0.1%</td>
<td>3</td>
<td>3.0</td>
<td>100.1%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Covtype</td>
<td>52.0%</td>
<td>51.6%</td>
<td>17</td>
<td>55.9</td>
<td>1156.6</td>
<td>360.4</td>
<td>-0.4%</td>
<td>-2</td>
<td>8.5</td>
<td>99.8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.5: Comparison of the effect of using diagonal covariances on accuracy (acc), number of components in the model (K), negative log-likelihood (-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.
6.6 Face Recognition Task

Previous sections showed that xokde++/diag obtains significant computational performance gains while retaining model quality. However, we have yet to test the high dimensional scenario. For this, we perform a face recognition task using the previously described L²F Face Database. We could use one of the crop setups directly, leaving us with feature vectors of 1024 to 4086 dimensions per sample, corresponding to the grayscale pixels of the image. However, a great part of this is redundant or unnecessary information. To avoid having large, yet uninformative feature vectors, which would lead to large amounts of training time, we selected patches of pixels spread throughout the image. This allows to have significantly lower redundancy, and feature dimensionality, while keeping information from the whole image. Figure 6.8 depicts the selected patches for training the models. These patches correspond to feature vectors of 128 dimensions, which is more than 4 times higher than the highest dimension dataset of the previous evaluation, and 8 times higher than the Letter dataset, which has a similar number of samples compared to the L²F Face Database.

Figure 6.8: Multiple examples of the masks effect on the extracted features over various face poses and expressions.

With this dataset, we were unable to use the oKDE implementation to train and test the model as the SVD computation did not converge, even with normalized features. Furthermore, we were also unable to complete training using the OISVM approach, since, in line with what happened with the covtype dataset, it was taking a long time to train.

To be able to compare our performance, we decided to train a batch SVM implemented using the WEKA Toolkit (Hall, Frank, Holmes, Pfahringer, Reutemann, and Witten 2009). The default complexity parameter $C=1$ was used. As the algorithm converges faster on normalized data, we used the default normalization which places the feature space in the $[-1, 1]$ range. This approach does not fit our requirements for incremental training, but has the advantage of providing a reliable upperbound regarding the expected accuracy.

In line with the literature review on face recognition and with preliminary experiments using batch GMMs, we decided to experiment with two sets of features: the grayscale pixels, equivalent to raw signal, and the half local binary patterns (LBPh) (Kang, Liao, Xiang, and Pan 2014). Table 6.6 summarizes the results: xokde++ achieves similar accuracy when compared to the batch SVM. It also shows that the LBPh feature set actually harms the classifier ability to model data. One explanation for this is that, since the LBP is a value between 0 and 15 (the binary pattern regarding the 4 lower adjacent pixels), the total feature variability will be lower and thus restricting the kernel estimate of all classes to the same area of the feature space. This
produces models that are similar to each other, and harms the classifier discriminative power. Another hint for this is the lower result also obtained by the SVM: this value indicates that the features, or at least the regions selected by the mask, are less discriminative than the grayscale pixels.

<table>
<thead>
<tr>
<th>Feature Set</th>
<th>xokde++</th>
<th>xokde++/diag</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixels</td>
<td>–</td>
<td>94.06%</td>
<td>95.37</td>
</tr>
<tr>
<td>LBPh</td>
<td>–</td>
<td>89.06%</td>
<td>82.41</td>
</tr>
</tbody>
</table>

Table 6.6: Classification accuracy for xokde++ and the batch SVM approach for the grayscale and LBPh feature sets.

6.7 Summary

In this chapter, we evaluated our implementation of the online kernel density estimation. To make a thorough comparison, we measured the performance under five different perspectives. The first two are intrinsic evaluations of model quality: the model average negative log-likelihood, and the average model complexity. Summarized in table 6.1 and in figures 6.1 and 6.2, we show that we achieve similar, even slightly better, performance than the original implementation. Next, in section 6.3, we measured the discriminative quality of the models by using them on a classification task. Again, we obtain better results than the original implementation that are also competitive against an online discriminative approach, as shown in table 6.3 and figures 6.3 and 6.5. Finally, in section 6.4, we measured time and memory requirements to train and test the models. In tables 6.3 and 6.4, and in figures 6.6 and 6.7, we show that our approach achieves major performance gains, requiring 90% less memory and being 3 to 10 times faster. For the diagonal approach of xokde++/diag, the time speedup is even greater, going from 11 to 40 times faster than oKDE.

With this analysis, we concluded our comparison with the state of the art work of oKDE, concluding that our approach achieves similar, sometimes slightly better, model quality, at a much lower computational cost. To cope with high dimensionality, and to keep the computational cost low, the best approach is to use diagonal matrices. As such, it is desirable to have a detailed comparison of the cost and benefits of using diagonal matrices. This was shown in section 6.5 and summarized in table 6.5. From this, we concluded that the diagonal approach had an average accuracy loss at around 1.35% absolute, while requiring 22% less memory and being 5.5 times faster.

Knowing that the diagonal approach has little accuracy loss, but is computationally more efficient, we experimented this approach with our high dimensional dataset, the L2F Face Database. Section 6.6 describes these experiments and shows that neither oKDE nor oISVM are able to train and test this dataset. Since none of these methods worked on the dataset, we used a batch SVM to have a comparison ground with our xokde++/diag performance. We achieve 94.1% accuracy, which is only 1.3% lower than the 95.4% achieved with a batch discriminative approach.
CHAPTER 6. RESULTS AND DISCUSSION
Conclusion

We defined a clear goal of estimating and maintaining a model through long operational times in the context of high dimensionality.

During the pursuit of this goal, five main challenges were identified: 1) How to avoid retraining the whole model as more samples arrive? 2) How to avoid keeping all previously observed samples? 3) How to deal with arbitrary data dimension and structure? 4) How to recover from numeric pitfalls, as the model needs to keep training even if by some reason something went wrong?

The first and second challenges were approached in chapters 2 and 3. This latter chapter, described the current state of the art to solve these first two challenges. Our work thus, starts by replicating this approach. However, this state of the art approach did not specifically address our third and fourth challenges, which were related with numeric stability and dimension scalability. In chapter 4, we presented our main contributions which mainly focus on solving or mitigating challenges 3) and 4). Chapter 4 also presented an additional contribution in the form of a new implementation that is easily extensible and parametrized, allowing, for example, to swap full covariance matrices by diagonal ones. This change, in addition to improving computational performance was later shown to also help numerical stability.

One of the main contributions of this work is a numerically robust and computationally efficient online kernel density estimator, able to cope with non-normalized data. Furthermore, our implementation is both extensible and easy to integrate in larger projects, due to its pure template library nature. Compared to the implementation used in the newest state of the art oKDE papers, our implementation is up to 40 times faster, needs 90% less memory, has greater numerical robustness, and produces, on average, models with higher quality.

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 (Huber 2009), and big data, where it now becomes possible to continuously capture and model high dimensional data samples using the KDE approach. It also provides the base work for future work on building complex semantic relationships from data (Mesnil, Rifai, Bordes, Glorot, Bengio, and Vincent 2015).

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 to improve computational performance include high performance computing adaptations, to make use of GPUs or co-Processors. Other alternatives include changing the hierarchical compression approach, to another, more paralelizable approach such as pairwise merging of components proposed by Runnalls (2007).
Still 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 (Liu, Lafferty, and Wasserman 2007; Krishnamurthy 2011; Azizyan, Singh, and Wasserman 2015).

Besides computational performance, model quality could also be improved. Here, instead of using full or diagonal covariance matrices, which are either too large for high dimensionality or too restrictive for certain non-linearly independent data, we could approximate the covariance with low rank perturbations (Magdon-Ismail and Purnell 2009). The authors claim this approach preserves important information, having similar quality of the full covariance, while presenting a lower computational cost. Another approach would be to experiment with Toeplitz matrices (Pasupathy and Damodar 1992; Cai, Ren, and Zhou 2014). However, using sparse or Toeplitz matrices may cause problems in the estimation of the log-determinant. Cai, Liang, and Zhou (2015) 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.

Regarding the compression phase, particularly on the Goldberger K-Means algorithm, other distance metrics could be experimented. Currently the algorithm uses the Kullback-Leibler divergence. Two other possible approaches would be the Bregman Divergence (Davis and Dhillon 2007) and the Hellinger distance.

Finally, another improvement on numeric stability and model quality may be obtained by avoiding correction of degenerate covariances, which always introduces some $\epsilon$ error. A solution is to use the degenerate Gaussian probability density function and computing the pseudo-inverse and pseudo-log-determinant (Mikheev 2006).


The covariance matrix of a random vector $X \in \mathbb{R}^n$ with mean vector $\mathbf{m}_x$ is defined as

$$C_x = E[(X - \mathbf{m})(X - \mathbf{m})^T]$$  \hspace{1cm} (A.1)

The $(i, j)^{th}$ element of this matrix $C_x$ is given by

$$C_{ij} = E[(X_i - m_i)(X_j - m_j)] = \sigma_{ij}$$  \hspace{1cm} (A.2)

The diagonal entries of the covariance matrix $C_x$ correspond to the variances of the dimensions of the random vector $X$, i.e.,

$$C_{ii} = E[(X_i - m_i)^2] = \sigma_i^2$$  \hspace{1cm} (A.3)

Since the diagonal entries are all positive, the trace of the matrix will also be positive

$$\text{Trace}(C_x) = \sum_{i=1}^{n} C_{ii} > 0$$  \hspace{1cm} (A.4)

The covariance matrix is symmetric, i.e., $C_x = C_x^T$ because

$$C_{ij} = \sigma_{ij} = \sigma_{ji} = C_{ji}$$  \hspace{1cm} (A.5)

The covariance matrix is also positive semidefinite. For $\mathbf{a} \in \mathbb{R}^n$

$$\mathbf{a}C_x \mathbf{a}^T \geq 0$$  \hspace{1cm} (A.6)

Since $C_x$ is symmetric, the product of its eigenvalues are all real and positive, and the eigenvectors are orthogonal

$$C_x = \mathbf{V} \Lambda \mathbf{V}^T = \sum_{i=1}^{n} \lambda_i \mathbf{v}_i \mathbf{v}_i^T$$  \hspace{1cm} (A.7)

As a consequence, the determinant of the covariance is non-negative,

$$\text{Det}(C_X) = \prod_{i=1}^{n} \lambda_i \geq 0$$  \hspace{1cm} (A.8)
Whitening Transformation

The whitening transform scales and rotates the data such that the resultant data dimensions will be uncorrelated and the variance in each dimension is unitary. As such, data corresponding to an ellipse when not whitened will correspond to a sphere after whitening. This process is sometimes called *sphering*.

Let \( x \) be a vector of zero-mean data. (fig B.1 (a) ) Its covariance matrix is

\[
\Sigma = E(xx^T)
\]  

(B.1)

If the data points in \( x \) are correlated, then their covariance \( \Sigma \) will not be a diagonal matrix. If \( \Sigma \) is positive definite, then it admits to the whitening transform and is reversible.

In order to de-correlate the data, we need to transform it so that the transformed data will have a diagonal covariance matrix. This transform can be found using the eigendecomposition.

\[
\Sigma \Phi = \Phi \Lambda
\]  

(B.2)

where \( \Lambda \) is a diagonal matrix and its diagonal elements are the eigenvalues. The columns of \( \Phi \) are the eigenvectors of the covariance matrix.

We can write the diagonalized covariance as:

\[
\Phi^T \Sigma \Phi = \Lambda
\]  

(B.3)

To apply the diagonalizing transform to a vector of data \( x \) we simply form:

\[
y = \Phi^T x
\]  

(B.4)

Thus, \( y \) is the de-correlated data (fig B.1 (b) ), meaning that its covariance \( E(yy^T) \) is now a diagonal matrix \( \Lambda \).

The eigenvalues in \( \Lambda \) may be the all equal or different. If we make them all the same, then we are *sphering* the data, as all dimensions will have the same scale, i.e. variance.

\[
\Lambda^{-1/2} \Lambda \Lambda^{-1/2} = I
\]  

(B.5)

Equivalently, substituting in eq. B.3

\[
\Lambda^{-1/2} \Phi^T \Sigma \Phi \Lambda^{-1/2} = I
\]  

(B.6)

To apply this whitening transform to \( y \) we can simply multiply it by this scale factor, obtaining the whitened data \( w \)

\[
w = \Lambda^{-1/2} y = \Lambda^{-1/2} \Phi^T x
\]  

(B.7)
Figure B.1: Example of a non whiten data (a), its decorrelated version (b), and its scaled version (c).

Now the covariance of $w$ is not only diagonal but also uniform (white), (fig B.1 (c)), since the covariance of $w$ is $E(ww^T) = I$

$$E(\Lambda^{-1/2}\Phi^T w w^T \Phi \Lambda^{-1/2}) = I \quad (B.8)$$

The inverse operation, i.e., the de-whitening transform, or colouring transform, $w^{-1} = x$ can be applied. First we scale the dimensions such that they reflect the desired variances along the principal axis. This changes the spherical contours into elliptic ones.

$$y = \Lambda^{1/2}w \quad (B.9)$$

Then, we rotate the dimensions such that they are now correlated.

$$x = \Phi y = \Phi \Lambda^{1/2}w \quad (B.10)$$