# Memory-Efficient Orthogonal Least Squares Kernel Density Estimation using Enhanced Empirical Cumulative Distribution Functions

**Martin Schafföner, Edin Andelic, Marcel Katz, Sven E. Krüger, Andreas Wendemuth**
Dept. of Electrical Engineering and Information Technology
Otto-von-Guericke-University
P. O. Box 4120, 39016 Magdeburg, Germany
martin.schaffoener@e-technik.uni-magdeburg.de

## Abstract

A novel training algorithm for sparse kernel density estimates by regression of the empirical cumulative density function (ECDF) is presented. It is shown how an overdetermined linear least-squares problem may be solved by a greedy forward selection procedure using updates of the orthogonal decomposition in an order-recursive manner. We also present a method for improving the accuracy of the estimated models which uses output-sensitive computation of the ECDF. Experiments show the superior performance of our proposed method compared to state-of-the-art density estimation methods such as Parzen windows, Gaussian Mixture Models, and $\epsilon$-Support Vector Density models [1].

## 1 Introduction

Estimation of probability density functions from data samples is a common problem in engineering and machine learning. Well known methods include (semi-)parametric models such as Gaussian mixture models (GMMs) estimated using the Expectation-Maximization (EM) procedure [5] and non-parametric models such as the Parzen windows method [6]. Semi-parametric models are relatively simple and computationally efficient but they require a number of parameters to be specified by the user, e. g. the number of mixture components and lower bounds on the individual Gaussians' covariances, which may be hard to obtain if no prior knowledge of the problem is available. Parzen windows do not suffer from these constraints, however, all the training samples are needed to define the density estimate.

Several approaches to estimate non-parametric sparse kernel density models employing regression on the distribution function have been suggested, including a support vector density ($\epsilon$-SVD) algorithm [1] using an epsilon-loss function, and a similar orthogonal least squares (OLS) algorithm with local regularization [4] employing Gram-Schmidt-orthogonalization and sparsification through forward regression. However, both of these methods are very memory-consuming during training time because they require the complete Gram matrix. To overcome this problem, we present an algorithm which constructs a sparse OLS density estimate using an order-recursive update of the orthogonal decomposition matrices by means of thin updates of the Gram matrix' pseudo-inverse which substantially reduces memory requirements.

The paper is organized as follows: In section 2, we revise the idea of density estimation as regression of the distribution function. Next, in section 3, we introduce some properties and problems of the empirical cumulative distribution function and possible solutions of these problems. In section 4, we lay out our algorithm in detail and compare them with other algorithms on a theoretic level, while section 5 gives experimental results and comparisons. The paper is concluded in section 6.

## 2 Kernel Density Estimation by Regression

A probability density $p(\boldsymbol{x})$ is defined as the solution of

$$\int_{-\infty}^{\boldsymbol{x}} p(\boldsymbol{t})\mathrm{d}\boldsymbol{t} = F(\boldsymbol{x}) \tag{1}$$

subject to the constraints

$$\int_{-\infty}^{\infty} p(\boldsymbol{u})\mathrm{d}\boldsymbol{u} = 1 \tag{2}$$

$$p(\boldsymbol{x}) \geq 0 \tag{3}$$

where $F(\boldsymbol{x})$ is the probability distribution function. Since $F(\boldsymbol{x})$ is unknown, (1) must be solved using an approximation $F_e(\boldsymbol{x})$, the empirical cumulative distribution function (cf. section 3).

Given an independent and identically distributed sample data set $\boldsymbol{X} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ with $\boldsymbol{x} \in \mathbb{R}^D$ drawn from the unknown distribution $F(\boldsymbol{x})$, we wish to estimate the unknown density $p(\boldsymbol{x})$ such that

$$p(\boldsymbol{x}) = \sum_{k=1}^{N} \alpha_k K(\boldsymbol{x}, \boldsymbol{x}_k) \tag{4}$$

subject to the constraints

$$\alpha_k \geq 0 \quad \forall k \tag{5}$$

$$\sum_{k=1}^{N} \alpha_k = 1 \tag{6}$$

using a kernel $K(\boldsymbol{x}, \boldsymbol{y})$ with the properties

$$K(\boldsymbol{x}, \boldsymbol{y}) \geq 0 \tag{7}$$

$$\int_{-\infty}^{x} K(\boldsymbol{t}, \boldsymbol{y}) \mathrm{d}\boldsymbol{t} \geq 0 \tag{8}$$

$$\int_{-\infty}^{\infty} K(\boldsymbol{t}, \boldsymbol{y}) \mathrm{d}\boldsymbol{t} = 1 \tag{9}$$

to satisfy the constraints from (2) and (3). Examples of kernels satisfying these constraints are

- the Gaussian kernel

$$K_{\text{gauss}}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{(2\pi\sigma^2)^{D/2}} \exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{y}||^2}{2\sigma^2}\right) \tag{10}$$

- the Epanechnikov kernel

$$K_{\text{epa}}(\boldsymbol{x}, \boldsymbol{y}) = \frac{3}{4}(1 - ||\boldsymbol{x} - \boldsymbol{y}||^2/\sigma^2)_+ \tag{11}$$

- the triangle kernel

$$K_{\text{tri}}(\boldsymbol{x}, \boldsymbol{y}) = (1 - ||\boldsymbol{x} - \boldsymbol{y}||/\sigma)_+ \tag{12}$$

with $\sigma > 0$.

A possible solution of (4) is provided by the well known method of Parzen windows [6] with $\alpha_k = 1/N, k = 1 \ldots N$ and any kernel (window function) satisfying constraints (7)–(9). This method is known to perform very well, however, it relies on all the problem's samples to characterize the solution. Therefore, we strive to obtain a solution with some or most $\alpha_k = 0$, i. e., a sparse approximation of the problem.

If the parameters of the kernel $K$ are considered fixed, the density model is completely characterized by the weight vector $\boldsymbol{\alpha}$. With (1), kernel density estimation can then be posed as the regression modeling problem

$$F_e(\boldsymbol{x}) = \sum_{k=1}^{N} \alpha_k q(\boldsymbol{x}, \boldsymbol{x}_k) + \epsilon(\boldsymbol{x}) \tag{13}$$

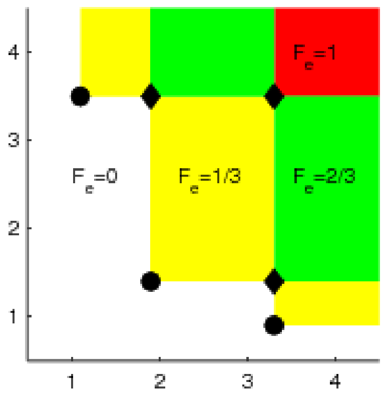

Figure 1: Construction of a two-dimensional example ECDF. Circles denote original problem samples, diamonds denote additional ECDF jumps, colored areas denote value of the ECDF.

subject to constraints (5) and (6) with

$$q(\boldsymbol{x}, \boldsymbol{x}_k) = \int_{-\infty}^{\boldsymbol{x}} K(\boldsymbol{t}, \boldsymbol{x}_k) \mathrm{d}\boldsymbol{t} \tag{14}$$

and $\epsilon(\boldsymbol{x})$ the modeling error at $\boldsymbol{x}$.

## 3 The Empirical Cumulative Distribution Function

The empirical cumulative distribution function is determined solely by the samples $\boldsymbol{x}_k, k = 1 \ldots N$ such that

$$F_e(\boldsymbol{x}) = \frac{\sum_{k=1}^{N} \Theta(\boldsymbol{x}, \boldsymbol{x}_k)}{N} \tag{15}$$

where $\Theta(\boldsymbol{x}, \boldsymbol{y})$ denotes point dominance

$$\Theta(\boldsymbol{x}, \boldsymbol{y}) = \{\boldsymbol{x} \succeq \boldsymbol{y}\} = \begin{cases} 1, & \text{if } \boldsymbol{x}_{(i)} \geq \boldsymbol{y}_{(i)} \forall i = 1 \ldots D \\ 0, & \text{else} \end{cases} \tag{16}$$

This means that $F_e(\boldsymbol{x})$ is a function which is given non-parametrically [5]. The ECDF has a staircase shape with $N$ jumps in the one-dimensional case. In figure 1 an admittedly degenerate two-dimensional problem with $N = 3$ problem sample points is shown where the corresponding ECDF has a much more complex shape with $6 > (N = 3)$ jumps.

From (15) it is not clear at which points the regression fit in (13) is to be evaluated. In the one-dimensional case, one usually samples $F_e$ at the problem's sample points $\boldsymbol{x}_k$ for convenience, since that's where the jumps of the function are located. In the multi-dimensional case, sampling $F_e$ at the problem's sample points does not capture the complexity of the ECDF. Even worse, in high dimensions a situation can occur where none of the samples dominates any other sample (as in figure 1), i. e., sampling $F_e$ at the problem sample points yields only the one function value $1/N$ instead of the entire range $[0, 1]$. Therefore, the ECDF has to be sampled at additional locations.

One possibility is to explicitly compute all the points at which jumps in the ECDF occur. This can be quite involving, but in [7] an algorithm is proposed which solves the problem in $\mathcal{O}(Dk)$ time where $k$ is the sample-dependent number of jumps of the ECDF. Since in high dimensions the number $k$ of output points may also be very large, one may resort to only using a feasible number of randomly picked points from this algorithm. This approach then will not capture the full complexity of the ECDF but at least use meaningful points which generate the entire range $[0, 1]$ of the ECDF.

## 4 Memory-Efficient Orthogonal Forward Regression

Consider that suitable locations $\boldsymbol{c}_s$, $s = 1 \dots S$ for checking the regression model (13) have been identified as in section 3, we rewrite (13) in a more convenient fashion:

$$\boldsymbol{F}_e = \boldsymbol{Q}\boldsymbol{\alpha} + \boldsymbol{\epsilon} \tag{17}$$

with $\boldsymbol{F}_{e(s)} = F_e(\boldsymbol{c}_s)$, $\boldsymbol{\alpha}_{(k)} = \alpha_k$, $\boldsymbol{Q}_{(s,k)} = q(\boldsymbol{c}_s, \boldsymbol{x}_k)$, $\boldsymbol{\epsilon}_{(s)} = \epsilon(\boldsymbol{c}_s)$. If an orthogonal decomposition of $\boldsymbol{Q} = \boldsymbol{W}\boldsymbol{A}$ with $\boldsymbol{w}_i^\top \boldsymbol{w}_j = 0, i \neq j$, is assumed, (17) can be written as

$$\boldsymbol{F}_e = \boldsymbol{W}\boldsymbol{g} + \boldsymbol{\epsilon} \tag{18}$$

with

$$\boldsymbol{g} = \boldsymbol{A}\boldsymbol{\alpha} \tag{19}$$

the weights in the orthogonal space $\boldsymbol{W}$. The optimal weight vector $\hat{\boldsymbol{g}}$ can be obtained as the solution of the least squares problem with local regularization

$$\hat{\boldsymbol{g}} = \text{argmin}_{\boldsymbol{g}} \boldsymbol{\epsilon}^\top \boldsymbol{\epsilon} + \sum_{k=1}^{N} \boldsymbol{\lambda}_{(k)} \boldsymbol{g}_{(k)}^2 \tag{20}$$

where $\boldsymbol{\lambda}$ is the regularization parameter vector. This vector is optimized based on the Bayesian evidence procedure [8]. Briefly, the update of the regularization parameter $\lambda_m$ works as follows:

$$\boldsymbol{\lambda}_{(m)}^{\text{new}} = \frac{\boldsymbol{\gamma}_{(m)} \boldsymbol{\epsilon}^\top \boldsymbol{\epsilon}}{(N - \boldsymbol{1}^\top \boldsymbol{\gamma}) \boldsymbol{g}_{(m)}^2} \tag{21}$$

where

$$\gamma_{(m)} = \frac{\boldsymbol{w}_m^\top \boldsymbol{w}_m}{\boldsymbol{\lambda}_{(m)}^{\text{old}} + \boldsymbol{w}_m^\top \boldsymbol{w}_m} \tag{22}$$

Details of the derivation of the formulas can be found in [4].

Inserting (18) into (20) yields

$$\hat{\boldsymbol{g}} = \text{argmin}_{\boldsymbol{g}} \boldsymbol{g}^\top \boldsymbol{W}^\top \boldsymbol{W} \boldsymbol{g} + \sum_{k=1}^{N} \boldsymbol{\lambda}_{(k)} \boldsymbol{g}_{(k)}^2 - 2 \boldsymbol{F}_e^\top \boldsymbol{W} \boldsymbol{g} \tag{23}$$

which reveals the merit of the orthogonal decomposition that the components of $\boldsymbol{g}$ can be optimized independently of each other which lends itself to forward selection if a sparse approximation of (23) is intended.

In [4] the modified Gram-Schmidt (MGS) procedure is proposed for the orthogonalization of $\boldsymbol{Q}$. However, since we are interested in a sparse approximation of (23) via forward selection, in the $m$th selection iteration we need to orthogonalize $N - m$ columns of $\boldsymbol{Q}$ onto the previously selected $m$ columns in $\boldsymbol{W}$. This implies that the complete matrix $\boldsymbol{Q}$ must be known and kept in memory. The memory complexity of the MGS algorithm is thus roughly $\mathcal{O}(SN + mN)$ for the complete $\boldsymbol{Q}$ matrix (including the orthogonalized parts here referred to as $\boldsymbol{W}$) and the $m$ complete rows of $\boldsymbol{A}$.

Therefore, we propose a more efficient algorithm which computes both $\boldsymbol{W}$ and $\boldsymbol{A}$ column-by-column without the need to keep unused columns of these matrices in memory [3]. In the $m$th forward selection step we consider the following partitioning of the reduced $\boldsymbol{Q}$ matrix and corresponding $\boldsymbol{\alpha}$

$$\boldsymbol{Q}_m = [\boldsymbol{Q}_{m-1} \boldsymbol{q}_m] \tag{24}$$

$$\boldsymbol{\alpha}_m = [\boldsymbol{\alpha}_{m-1} \alpha_m]^\top \tag{25}$$

such that the square loss from (20) becomes

$$L(\boldsymbol{\alpha}_{m-1}, \alpha_m) = ||\boldsymbol{Q}_{m-1} \boldsymbol{\alpha}_{m-1} - (\boldsymbol{F}_e - \boldsymbol{q}_m \alpha_m)||^2 \tag{26}$$

The minimum of (26) is given by

$$\hat{\boldsymbol{\alpha}}_{m-1} = \boldsymbol{Q}_{m-1}^\dagger (\boldsymbol{F}_e - \boldsymbol{q}_m \alpha_m) \tag{27}$$

with $\boldsymbol{Q}_{m-1}^\dagger$ the pseudo-inverse of $\boldsymbol{Q}$. This yields after insertion into (26)

$$\begin{aligned} L(\alpha_m) = ||&(\boldsymbol{I} - \boldsymbol{Q}_{m-1} \boldsymbol{Q}_{m-1}^\dagger) \boldsymbol{q}_m \alpha_m \\ &- (\boldsymbol{I} - \boldsymbol{Q}_{m-1} \boldsymbol{Q}_{m-1}^\dagger) \boldsymbol{F}_e||^2 \end{aligned} \tag{28}$$

with $\boldsymbol{I}$ the identity matrix of appropriate size.

The minimum of (28) is reached at

$$\alpha_m = \boldsymbol{w}_m^\dagger (\boldsymbol{I} - \boldsymbol{Q}_{m-1} \boldsymbol{Q}_{m-1}^\dagger) \boldsymbol{F}_e \tag{29}$$

Considering that the pseudo-inverse of $\boldsymbol{w}$ is given by

$$\boldsymbol{w}_m^\dagger = \frac{\boldsymbol{w}_m^\top}{||\boldsymbol{w}_m||^2} \tag{30}$$

(29) may be written as

$$\begin{aligned}\alpha_m &= \frac{\boldsymbol{w}_m^\top(\boldsymbol{I}-\boldsymbol{Q}_{m-1}\boldsymbol{Q}_{m-1}^\dagger)\boldsymbol{F}_e}{||\boldsymbol{w}_m||^2} \\ &= \frac{\boldsymbol{q}_m^\top(\boldsymbol{I}-\boldsymbol{Q}_{m-1}\boldsymbol{Q}_{m-1}^\dagger)^\top(\boldsymbol{I}-\boldsymbol{Q}_{m-1}\boldsymbol{Q}_{m-1}^\dagger)\boldsymbol{F}_e}{||\boldsymbol{w}_m||^2}\end{aligned} \tag{31}$$

The matrix

$$\boldsymbol{P}_m = \boldsymbol{I}-\boldsymbol{Q}_{m-1}\boldsymbol{Q}_{m-1}^\dagger \tag{32}$$

is an orthogonal projection matrix which implies it being symmetric and idempotent. Thus, (31) can be simplified as

$$\alpha_m = \boldsymbol{w}_m^\dagger\boldsymbol{F}_e \tag{33}$$

Combining (33) with (27), the weight vector $\hat{\boldsymbol{\alpha}}_m$ may be updated as

$$\hat{\boldsymbol{\alpha}}_m = \begin{bmatrix}\hat{\boldsymbol{\alpha}}_{m-1}\\ \hat{\alpha}_m\end{bmatrix} = \begin{bmatrix}\boldsymbol{Q}_{m-1}^\dagger - \boldsymbol{Q}_{m-1}^\dagger\boldsymbol{q}_m\boldsymbol{w}_m^\dagger\\ \boldsymbol{w}_m^\dagger\end{bmatrix}\boldsymbol{F}_e \tag{34}$$

yielding the update

$$\boldsymbol{Q}_m^\dagger = \begin{bmatrix}\boldsymbol{Q}_{m-1}^\dagger - \boldsymbol{Q}_{m-1}^\dagger\boldsymbol{q}_m\boldsymbol{w}_m^\dagger\\ \boldsymbol{w}_m^\dagger\end{bmatrix} \tag{35}$$

of the current pseudo-inverse.

Because every projection $\boldsymbol{w}_m = \boldsymbol{P}_m\boldsymbol{q}_m$ lies in a subspace orthogonal to $\boldsymbol{Q}_{m-1}$, it follows directly that $\boldsymbol{w}_i^\top\boldsymbol{w}_j = 0, i \neq j$. Therefore, the orthogonal decomposition of $\boldsymbol{Q}_m$ can be updated as

$$\boldsymbol{W}_m = [\boldsymbol{W}_{m-1}\boldsymbol{w}_m] \tag{36}$$

$$\boldsymbol{A}_m = \begin{bmatrix}\boldsymbol{A}_{m-1}\\ \boldsymbol{0}_{m-1}^\top\end{bmatrix}(\boldsymbol{W}_m^\top\boldsymbol{W}_m)^{-1}\boldsymbol{W}_m^\top\boldsymbol{q}_m \tag{37}$$

for which we call our algorithm order-recursive orthogonal least squares (OROLS). The inversion of $(\boldsymbol{W}_m^\top\boldsymbol{W}_m)$ is trivial since it is diagonal. It is important to monitor the condition number of $\boldsymbol{W}_m$ as it increases as the number $m$ of admitted columns grows, so to ensure numerical stability a termination threshold on the condition number must be defined.

So the complete algorithm is as follows:

1. Define the set of $\mathcal{M} = \{1,\ldots,N\}$ of all possible sample indices, the set $\mathcal{S} = \emptyset$ of already admitted sample indices, the current solution index $m = 1$

2. For each $n \in \mathcal{M}\backslash\mathcal{S}$, do the following
   (a) Compute the corresponding updates of $\boldsymbol{W}_m$ and $\boldsymbol{A}_m$ via (36) and (37)
   (b) If the condition number $\boldsymbol{w}_m^\top\boldsymbol{w}_m$ violates a threshold, continue with the next $n$
   (c) Solve the LS-problem from (23) while optimizing the regularization parameter $\boldsymbol{\lambda}_{(m)}$
   (d) Reconstruct the original weights $\boldsymbol{\alpha}_m = \boldsymbol{A}_m^{-1}\boldsymbol{g}_m$. If they contain any negative elements (which would violate the constraint from (5)), continue with the next $n$
   (e) Determine the fitness $l_n$ of the current sample by computing its leave-one-out (LOO) test error [4]

3. Admit the optimal sample $\hat{n} = \text{argmin}_n l_n$ into $\mathcal{S}$, re-compute the corresponding updates of $\boldsymbol{W}_m$ and $\boldsymbol{A}_m$ and continue with the next $m$ if $l_{\hat{n}}$ decreases the previous $m$-iteration's LOO-score, otherwise terminate

4. Normalize the original weights to meet (6)

The memory requirement for this algorithm is $\mathcal{O}(3Sm + m^2/2)$ for the candidate $\boldsymbol{W}_m$, $\boldsymbol{A}_m$ and $\boldsymbol{Q}_m^\dagger$matrices needed in step 2 and the old $\boldsymbol{Q}_{m-1}^\dagger$ matrix which needs to be saved until the optimal candidate has been chosen in step 3. If $m < N/3$ (which we usually strive for), this compares quite favorable with the MGS-algorithm introduced in [4] whose memory requirements are $\mathcal{O}(SN + mN)$ and with the support-vector method presented in [1] whose memory requirements are $\mathcal{O}(SN + N^2)$.

## 5 Experiments

To assess the performance of our proposed algorithm, experiments on three problems were performed. We compared the proposed method with classical density estimation procedures, i. e., Gaussian Mixture Models and Parzen windows, and the $\epsilon$-SVD method. Throughout all experiments, the Gaussian kernel (10) with integral

$$\begin{aligned}q(\boldsymbol{x},\boldsymbol{y}) &= \int_{-\infty}^{\boldsymbol{x}} K_{\text{gauss}}(\boldsymbol{t},\boldsymbol{y})\mathrm{d}\boldsymbol{t} \\ &= \prod_{d=1}^{D}\left(1+\operatorname{erf}\left(\frac{||\boldsymbol{x}_{(d)}-\boldsymbol{y}_{(d)}||}{\sqrt{2}\sigma}\right)\right)\end{aligned} \tag{38}$$

was used. This kernel was chosen because it additionally satisfies Mercer's conditions [9] which enables comparison of our method with the $\epsilon$-SVD procedure. It was assumed that all kernels in a model share the same covariance $\sigma^2$.

GMMs with diagonal covariance matrices were trained using the EM-algorithm. Initialization of the EM-algorithm was performed using k-means clustering, which in turn had been randomly seeded. $\epsilon$-SVD models and the proposed OROLS density models were trained using naive ECDF sampling (ECDF only sampled at the problem's sample points) and enhanced ECDF sampling (ECDF sampled at additional jump points, cf. section 3) schemes. For the $\epsilon$-SVD method no model selection was required as all the parameters including the kernel width are set automatically. Model selection procedures for the other methods are discussed individually for each data set.

## Artificial 2D-Problem

The first one is an artificial problem already considered in [1] were 100 training sets of 60 samples each and a test set of 10000 samples are generated from the following density

$$\begin{aligned} p(x,y) =& \frac{1}{4\pi}\exp\left(-\frac{(x-2)^2+(y-2)^2}{2}\right) \\ &+\frac{0.35}{8}\exp(-0.7|x+2|-0.5|y+2|) \end{aligned} \tag{39}$$

As a baseline, GMMs with 2 and 4 centers were trained as described above. Parzen windows setup was straightforward. Next, $\epsilon$-SVD models were trained using both naive ECDF sampling and enhanced ECDF sampling. Finally, the proposed algorithm was used to estimate another set of density models, again using both incarnations of ECDF sampling. For all the 7 sets of 100 density models the $L_1$-norm of the test errors $\boldsymbol{e}$ were computed on the 10000 sample test set. Selection of the suitable kernel width for Parzen windows and the OROLS models was performed by selecting the kernel width which produces the lowest average $L_1(\boldsymbol{e})$ over all 100 training sets.

Figure 2 compares the achievements of the different algorithms showing clearly that the proposed algorithm performs superior to all other algorithms if used with enhanced ECDF sampling and at least comparable with the GMMs if used with naive ECDF sampling. It can also be seen clearly that the proposed algorithm provides better results than the $\epsilon$-SVD method, regardless of the ECDF sampling used. Table 1 summarizes some details of the trained models. It also supports the superiority of our proposed algorithm: While it can be observed that, in general, using enhanced ECDF sampling results in a larger number $m$ of samples in the solution, this number does not increase as much for the OROLS algorithm as for the $\epsilon$-SVD algorithm, yet much more accurate models are achieved using the OROLS algorithm. Furthermore, it can be seen that as, on average, the OROLS algorithm selects about 22% of the problem samples using the naive ECDF, and about 29% of the problem samples using the enhanced ECDF, the computation of

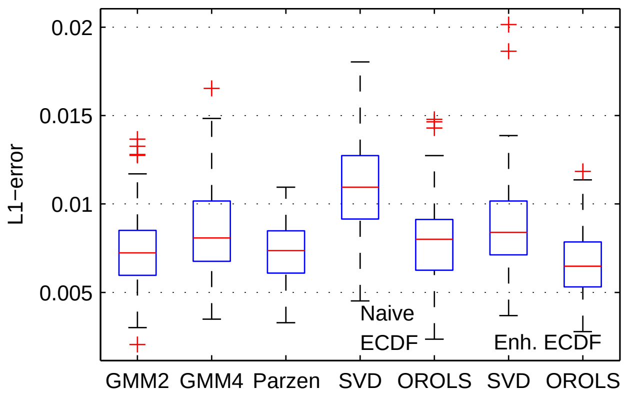

Figure 2: Comparison of the $L_1$-norm of the errors committed by different density estimation algorithms on the artificial 2D-problem

| Model | ECDF | Kernel Width | # centers |
|---|---|---|---|
| Parzen | — | 0.6 | 100 |
| $\epsilon$-SVD | Naive | 1.34(±0.27) | 13.1(±3.7) |
| $\epsilon$-SVD | Enhanced | 0.82(±0.26) | 26.1(±9.1) |
| OROLS | Naive | 1.0 | 13.4(±2.3) |
| OROLS | Enhanced | 1.0 | 17.6(±3.3) |

Table 1: Details of the models for the artificial 2D-problem.

the solution using OROLS is less memory-consuming than $\epsilon$-SVD and MGS-OLS based density estimation procedures, cf. section 4.

## Ripley data set

This second experiment is based on a data set considered in [10]. It consists of two classes in two dimensions with 125 training samples and 500 testing samples each. We trained GMMs with 2 and 4 components, Parzen windows, and $\epsilon$-SVD and OROLS models, each of the latter two with naive and enhanced ECDF sampling. We then assessed the classification error on the test set. As in the previous experiment, no parameter needed to be manually tuned for the $\epsilon$-SVD algorithm, while the kernel width for Parzen windows and OROLS models were set according to test error. Assignment of a test sample $\boldsymbol{x}$ to a class $C \in \{-1,+1\}$ was performed using maximum a-posteriori and Bayes' rule:

$$\begin{aligned} \hat{C} = \underset{C}{\operatorname{argmax}}\, P(C|\boldsymbol{x}) &= \underset{C}{\operatorname{argmax}} \frac{p(\boldsymbol{x}|C)P(C)}{p(\boldsymbol{x})} \\ &= \underset{C}{\operatorname{argmax}}\, p(\boldsymbol{x}|C)P(C) \end{aligned} \tag{40}$$

where, in this case, $P(C=+1) = P(C=-1) = 0.5$.

The results of the experiment can be seen in ta-

| Model | ECDF | $\sigma$ | # centers | err. |
|---|---|---|---|---|
| GMM2 | — | — | 2 / 2 | 9.1% |
| GMM4 | — | — | 4 / 4 | 9.2% |
| Parzen | — | 0.28 | 125 / 125 | 8.1% |
| $\epsilon$-SVD | Naive | 0.157 | 17 / 12 | 10.0% |
| $\epsilon$-SVD | Enhanced | 0.075 | 77 / 51 | 9.8% |
| OROLS | Naive | 0.17 | 13 / 6 | 8.9% |
| OROLS | Enhanced | 0.25 | 7 / 7 | 8.5% |

Table 2: Details and classification error rates of the density models on the Ripley data set. The number of solution centers is reported individually for each class.

ble 2. It can easily be seen that our proposed OROLS-algorithm has a lower test error than the $\epsilon$-SVD-algorithm while needing considerably fewer samples in the solution. On the other hand, it can be observed that using enhanced ECDF sampling for the training of the models improves the test error regardless of the training algorithm. In [11] it is reported that the theoretical Bayes error rate of this problem is about 8%, while the paper reports error rates of 10.6% for standard discriminatory SVM with Gaussian kernel and 38 solution vectors, and an error rate of 9.3% for the relevance vector machine with Gaussian kernel and four solution vectors. Compared to the previous artificial 2D-problem, the advantage of the OROLS-algorithm with respect to memory requirements is even more striking: In the worst case, 13 samples are selected, which is about 1/10 of the samples — much less than the rough 1/3 of the samples which would make competing $\epsilon$-SVD and MGS-OLS algorithms break even with the OROLS algorithm.

**Thyroid data set**

The thyroid disease data set is part of the UCI repository of machine learning problems [12]. It was thoroughly studied, e. g., in [13]. It is a two-class problem with 5-dimensional feature vectors. For our experiments, the problem setting from [13] was used, where 100 realizations of training sets with 140 samples each and test sets of 75 samples each had been generated. The 140 training samples of each training set consisted of roughly 40 positively labeled and 100 negatively labeled samples.

In contrast to the previous experiments, model selection, i. e., selection of the number of components of the GMMs and the kernel width for Parzen windows and OROLS models, was performed using thourough cross-validation similar to the procedure described in [13]. Models with varying parameters were trained on each of the first 5 training sets. The models' performance was assessed on the 5 cross-validation test sets which were concatenations of the 4 respective unused training sets. The parameters for the complete evaluation of the 100 realizations were then determined by averaging the parameters leading to the best results on the cross-validation sets.

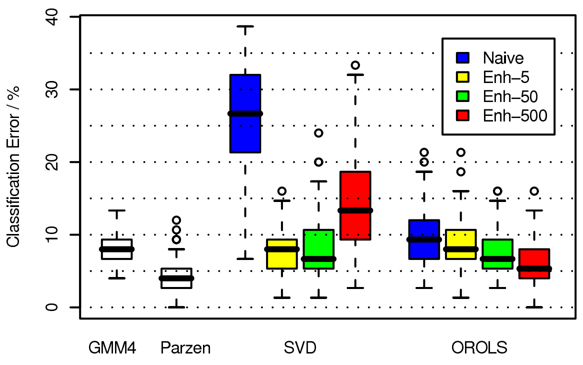

Figure 3: Classification errors on the Thyroid data set performed by different probability density models using different samplings from the ECDF for training.

The use of the enhanced ECDF sampling scheme was modified for this experiment. As stated in section 3, the number of jumps of the ECDF is problem-dependent, but usually much larger than the number of problem samples. For the problem at hand, the about 100 negatively labeled training samples would have generated an ECDF with roughly 1.5 million jumps. Instead of using all of these points, we conducted experiments where a maximum of 5, 50, and 500 random points was chosen from each level of the ECDF; we denote these experiments with Enh-5, Enh-50, and Enh-500, respectively.

Classification of the test samples was also performed using (40) with the class' priors estimated from the relative frequency of the labels in the training set. Figure 3 compares GMMs with 4 components and Parzen windows with $\epsilon$-SVD and OROLS models trained using different ECDF sampling schemes, as described above. It can be seen that the accuracy of the OROLS models improves as the number of ECDF samples increases, which supports the idea that naive sampling as well as under-sampling the enhanced ECDF significantly degrades the performance of the so-estimated density models. If we consider a maximum of 500 samples per ECDF level sufficient then it shows that the sparse OROLS kernel density models greatly outperform the best GMM models; however, the accuracy of the Parzen windows model is not met. Furthermore, OROLS models again outperform $\epsilon$-SVD models in all cases while producing sparser models. Through the obtained sparsity the OROLS algorithm takes advantage of its lower memory requirements compared to $\epsilon$-SVD as less than one third of the samples is used for the solutions.

| Model | ECDF | Kernel Width | # centers |
|---|---|---|---|
| GMM4 | — | — | 4 / 4 |
| Parzen | — | 0.27 | $\approx 100$ / $\approx 40$ |
| $\epsilon$-SVD | Naive | $2.85 \pm 1.51$ | 43.5 / 9.2 |
| $\epsilon$-SVD | Enh-5 | $0.067 \pm 0.015$ | 35.2 / 34.6 |
| $\epsilon$-SVD | Enh-50 | $0.052 \pm 0.014$ | 52.3 / 38.0 |
| $\epsilon$-SVD | Enh-500 | $0.031 \pm 0.01$ | 81.6 / 40.0 |
| OROLS | Naive | 0.17 | 12.9 / 10.6 |
| OROLS | Enh-5 | 0.07 | 16.1 / 12.9 |
| OROLS | Enh-50 | 0.07 | 17.7 / 13.4 |
| OROLS | Enh-500 | 0.07 | 18.1 / 14.6 |

Table 3: Details of the kernel density models on the Thyroid data set. The number of centers is reported individually for each class.

It might be surprising at first to see the performance of the $\epsilon$-SVD decrease (after an initial improvement over naive ECDF sampling) with increasing number of ECDF samples, especially in the Enh-500 setting. This can be attributed to the automatic tuning of the kernel width. The kernel width selected is the largest feasible, however, it still results in overfitted $\epsilon$-SVD models.

Our algorithm also compares favorable to other methods, even if performance is not en par with discriminative models such as SVM (classification error $4.8 \pm 2.2\%$) or kernel fisher discriminant (classification error $4.2 \pm 2.1\%$) [13]. On the other hand, the well-known C4.5 algorithm performs much worse for classification with an error of about 10.2% [14].

## 6 Conclusion

We have presented an algorithm which constructs sparse probability density models from training sample in a memory-efficient and accurate way. It outperforms standard, conventional models like EM-trained Gaussian mixture models while still producing sparse models. We have also shown the importance of sensible application of the ECDF in the training process in that a sampling at points additional to the original problem samples impressively improves the accuracy of the so-trained models.

In the future we will focus on extending the application of this algorithm to larger dimensions and to larger data sets in general, e. g. speech data.

## Acknowledgments

Martin Schafföner's work was partially funded by a grant from the Friedrich-Naumann-Foundation.

## References

[1] V. N. Vapnik and S. Mukherjee, “Support vector method for multivariate density estimation,” in *Advances in Neural Information Processing Systems 12*, S. A. Solla, T. K. Leen, and K.-R. Müller, Eds., pp. 659–665. MIT Press, 2000.

[2] E. Andelic, M. Schafföner, M. Katz, S. E. Krüger, and A. Wendemuth, “Kernel least squares models using updates of the pseudoinverse,” *Neural Computation*, vol. 18, no. 12, dec 2006, to appear.

[3] E. Andelic, M. Schafföner, M. Katz, S. E. Krüger, and A. Wendemuth, “Updates for nonlinear discriminants,” in *Proc. Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 2007.

[4] S. Chen, X. Hong, and C. J. Harris, “Sparse kernel density construction using orthogonal forward regression with leave-one-out test score and local regularization,” *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 34, no. 4, pp. 1708–1717, aug 2004.

[5] D. W. Scott, *Multivariate density estimation: theory, practice, and visualization*, Wiley series in probability and mathematical statistics. John Wiley & Sons, Inc., 1992.

[6] E. Parzen, “On estimation of a probability density function and mode,” *Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[7] C. M. Fonseca, “Output-sensitive computation of the multivariate ecdf and related problems,” in *COMPSTAT 2002 - Proceedings in Computational Statistics*, W. Härdle and B. Rönz, Eds. 2002, Springer.

[8] D. J. C. MacKay, “Bayesian interpolation,” *Neural Computation*, vol. 4, no. 3, pp. 415–447, 1992.

[9] J. Mercer, “Functions of positive and negative type and their connection with the theory of integral equations,” *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, vol. 209, pp. 415–446, 1909.

[10] B. D. Ripley, *Pattern Recognition and Neural Networks*, Cambridge University Press, Cambridge, UK, 1996.

[11] M. E. Tipping, “Sparse bayesian learning and the relevance vector machine,” *Journal of Machine Learning Research*, vol. 1, pp. 211–244, jun 2001.

[12] "UCI ML repository content summary," URL: http://www.ics.uci.edu/ mlearn/MLSummary.html.

[13] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller, "Fisher discriminant analysis with kernels," *Neural Networks for Signal Processing*, vol. 9, pp. 41–48, 1999.

[14] G. I. Webb, "Further experimental evidence against the utility of occam's razor," *Journal of Artificial Intelligence research*, vol. 4, pp. 397–417, 1996.