Carnegie Mellon University

Dietrich College of Humanities and Social Sciences Dissertation

Submitted in Partial Fulfillment of the Requirements

For the Degree of Doctor of Philosophy

Title: Adaptive Efficient Histograms

Presented by: Xiaoyi Gu

Accepted by: Department of Statistics & Data Science

Readers:

ALESSANDRO RINALDO, ADVISOR	DATE
ARUN KUMAR KUCHIBHOTLA, ADVISOR	DATE
LEMAN AKOGLU	DATE
MIKAEL KUUSELA	DATE
JAMES SHARPNACK	DATE
LARRY WASSERMAN	DATE

Approved by the Committee on Graduate Degrees:

RICHARD SCHEINES, DEAN	DATE

Adaptive Efficient Histograms

Xiaoyi Gu

December 16, 2021

A dissertation submitted in partial fulfillment
of the requirements for the Degree of Doctor of Philosophy

Department of Statistics & Data Science
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213

Thesis Committee:
Alessandro Rinaldo, Chair
Arun Kumar Kuchibhotla, Chair
Leman Akoglu
Mikael Kuusela
Larry Wasserman
James Sharpnack (University of California, Davis)

To my families and friends.

Acknowledgements

First and foremost, I would like to thank my advisors, Alessandro Rinaldo and Arun Kuchibhotla. The success of this dissertation would not have happened without their continuous support and guidance. Not only did they offer endless valuable research ideas and suggestions throughout this entire process, they are also extremely supportive and understanding when I was struggling, especially during a global pandemic. Research has always been full of challenges. Although not all of the directions we attempted led to successful results, what I learned from Ale and Arun during the process is invaluable. Their level of enthusiasm has inspired me to keep pursuing challenging and interesting problems. I would also like to thank the rest of my committee members — Larry Wasserman, Leman Akoglu, James Sharpnack, and Mikael Kuusela, for their helpful comments and suggestions.

My gratitude also goes to all my colleagues and friends over the past five years. I first would like to thank Yufei, Yue and Zongge, who made my experience at CMU much more enjoyable and positive. I would also like to thank other members of our Chinese group — Minshi, Xuran, Boyan, Xiaoyi, Jinjin, Shengming, Jining, Lingxue, Wanshan and Darren, for our cheerful conversations and the help and support we have offered each other. Many thanks to my other fellow PhD students who I have shared so many fond memories with, including Maria, Ben, Ciaran, Matteo, Tudor, Manjari,

Alan, Robin, Nic, Kayla, Neil, Kwangho, Shamindra, Ilmum, and JaeHyeok.

Finally, I'd like to take the opportunity to thank my parents for their unconditional love and encouragement, and my partner Hans for his companionship and support. I love you.

Abstract

Nonparametric density estimation is a fundamental task in statistics. In many applications, such as clustering, non-parametric testing, classification, anomaly detection and topological data analysis, density estimation is performed as a necessary preliminary step to solve more complex problems. Unfortunately, both the theoretical guarantees and the computational costs of virtually all commonly used density estimators are impacted significantly by the dimensionality of the data. This is due to the intrinsic hardness of the problem: minimax optimal density estimation is computationally infeasible even in small dimensions. In order to overcome the computational and theoretical limitations of the classical density estimations framework, in this thesis we study computationally efficient methods for obtaining adaptive histograms, piecewise constant density estimators over data adaptive partitions defined by axis-aligned hyperrectangles.

We consider a variant of the density estimator tree (DET) method of Ram and Gray (2011), a very fast, fully data-driven greedy CART-like procedure that returns a tree-structured density estimator. We show through extensive simulations that our modified DET procedure outperforms the standard version of DET, as well as traditional density estimators under a variety of scenarios and metrics, and is highly interpretable.

Inference for classical density estimation methods comes at a price, generally requiring certain smoothness conditions on the underlying density and computationally intensive procedures such as the bootstrap. In addition, the greedy nature of the DET method and the lack of theoretical understanding of its properties makes the task even more formidable. The second contribution of this thesis is the development of computationally feasible procedures to construct confidence sets for high-dimensional distributions based on the DET estimator. Our methodology relies on sample splitting and harness the explicit form of the upper level sets of the DET estimations, which can be easily evaluated as a union of axes-aligned hyper-rectangles. The confidence sets we develop come with coverage guarantees that are finite-sample, dimension free and do not require any smoothness on the data generating distribution. We propose three ways for building such confidence sets, and provide efficient algorithms that return density functions guaranteed to belong to the confidence set. We explore various applications of our methods, including classification, clustering, anomaly detection, and background subtraction.

Contents

Abstract v

Contents vii

List of Figures x

List of Tables xii

1 Introduction 1

1.1 Literature 2

1.1.1 Classical Density Estimators 2

1.1.2 Recent Advances 4

1.2 Contributions 7

2 Density Estimation Trees 9

2.1 DET with L_2 loss 9

2.1.1 Problem Formulation 10

2.1.2 Algorithm 11

2.1.3 Computational Complexity 14

2.2 DET with Maximum Likelihood 16

2.2.1 Problem Formulation 16
2.2.2 Algorithm 18
2.2.3 Computational Complexity 18
2.3 Bagged DET 18
2.4 Summary 18

3 Density Estimation with Confidence 21
3.1 DET with Confidence 21
3.1.1 VC-based Confidence Set 24
3.1.2 Empirical Process Based Confidence Set 25
3.1.3 Multiscale Likelihood Ratio Tests 28
3.1.4 Algorithm 30
3.2 Optimization Methods 32
3.2.1 Constrained MLE 34
3.2.2 Constrained Maximum Entropy 34
3.3 Summary 38

4 Numerical Experiments 41
4.1 DET(L_2) and DET(MLE) 41
4.1.1 Accuracy and Speed 42
4.1.2 Adaptability 44
4.1.3 Interpretability 53
4.2 Bagging 56
4.3 DET with Confidence 57
4.3.1 Accuracy and Speed 58
4.3.2 Adaptability 60

4.4 Summary 63

5 Applications **67**

5.1 Level Set Trees 67

5.2 Background Subtraction 70

5.3 Anomaly Detection 72

5.4 Classification 74

5.5 Summary 77

6 Software **79**

6.1 Installation 79

6.2 Functions 80

6.2.1 `det` 80

6.2.2 `det_cf` 82

Bibliography **85**

List of Figures

4.1 Density Estimates for KDE, DET(L_2),DET(MLE), OPT,Essential Histogram, and FDE when the underlying distribution is a univariate claw distribution. Sample size is 5000. 46

4.2 Density Estimates for KDE, DET(L_2),DET(MLE), OPT,Essential Histogram, and FDE when the underlying distribution is a univariate harp distribution. Sample size is 5000. 46

4.3 Density Estimates for KDE, DET(L_2),DET(MLE), OPT,Essential Histogram, and FDE when the underlying distribution is a univariate piecewise constant distribution with 10 discontinuities. Sample size is 5000. 48

4.4 Density Estimates for KDE, DET(L_2), and DET(MLE) when the underlying distribution is a mixture of two gaussian distributions with the same covariance matrix. Sample size is 5000. 49

4.5 Density Estimates for KDE, DET(L_2), and DET(MLE) when the underlying distribution is a mixture of two gaussian distributions with different covariance matrices. Sample size is 5000. 50

4.6 Density Estimates for KDE, DET(L_2), and DET(MLE) when the underlying distribution is a piecewise constant distribution, with 2 splits on the x-axis and 4 splits on the y axis. Sample size is 5000. 51

4.7 Density Estimates for KDE, DET(L_2), and DET(MLE) when the underlying distribution is a piecewise constant distribution, with 3 splits on the x-axis and 0 splits on the y axis. Sample size is 5000. 52

4.8 Contourplot of the density Estimates for KDE, DET(L_2), and DET(MLE) when the underlying distribution is a uniform density supported on two intertwined moon shaped domains. 54

4.9 Contourplot of the density Estimates for KDE, DET(L_2), and DET(MLE) when the underlying distribution is a uniform density supported on two concentric circles. 55

4.10 Density Estimates for DET(MLE), DET-CF(VC),DET-CF(EP), DET-CF(MLR), DET-CF(CM), and DET-CF(MaxEnt) when the underlying distribution is a univariate claw distribution. Sample size is 5000. 62

4.11 Density Estimates for DET(MLE), DET-CF(VC),DET-CF(EP), DET-CF(MLR), DET-CF(CM), and DET-CF(MaxEnt) when the underlying distribution is a univariate harp distribution. Sample size is 5000. . . . 62

4.12 Density Estimates for DET(MLE), DET-CF(VC),DET-CF(EP), DET-CF(MLR), DET-CF(CM), and DET-CF(MaxEnt) when the underlying distribution is a univariate piecewise constant distribution with 10 discontinuities. Sample size is 5000. 65

5.1 Demonstration of the Level Set Tree for DET(MLE) on (a) univariate claw distribution (b) univariate harp distribution. Figure on the left show s the underlying density function and the DET(MLE) estimates. Figure on the right plots the level set tree. 69

5.2 Application in background Subtraction: comparison of DET(MLE) and KDE. 71

List of Tables

2.1 Training and Querying computational complexity of KDE, tree-based KDE, OPT, LL-OPT, and DET. 15

4.1 Error in Hellinger Distance between the true density and KDE, OPT, DET(L_2), and DET(MLE). The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a mixture of four Gaussian distributions. 43

4.2 Average CPU time in seconds of KDE, OPT, DET(L_2), and DET(MLE). The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a mixture of four Gaussian distributions. . . . 44

4.3 Error in Hellinger Distance between the true density and KDE, OPT, DET(L_2), and DET(MLE). The numbers in parentheses are standard errors from 20 replicas.The underlying distribution is a product of (independent) piecewise constant densities with 10 discontinuities along each dimension. 45

4.4 Average CPU time in seconds of KDE, OPT, DET(L_2), and DET(MLE). The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a product of (independent) piecewise constant densities with 10 discontinuities along each dimension. 47

4.5 Variable importance estimation of each covariate by DET(L_2) and DET(MLE) when the underlying distribution is $X \in \mathbb{R}^{10}$, where $(X_1, X_2, X_3) \sim \mathcal{N}(\mu, \Sigma)$ for some μ and Σ, and $(X_4, \ldots, X_{10}) \sim \text{Unif}(0, 1)$ 56

4.6 Error in Hellinger Distance between the true density and DET and Bagged DET. The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a mixture of four Gaussian distributions. 57

4.7 Error in Hellinger Distance between the true density and DET, Bagged DET. The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a multivariate piecewise constant distribution. 58

4.8 Error in Hellinger Distance between the true density and DET-CF(VC), DET-CF(EP), DET-CF(MLR), DET-CF(CM), and DET-CF(MaxEnt). The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a mixture of four Gaussian distributions. . . . 60

4.9 Average CPU time in seconds of DET-CF(VC), DET-CF(EP), DET-CF(MLR), DET-CF(CM), and DET-CF(MaxEnt). The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a mixture of four Gaussian distributions. 61

4.10 Error in Hellinger Distance between the true density and DET-CF(VC), DET-CF(EP), DET-CF(MLR), DET-CF(CM), and DET-CF(MaxEnt). The numbers in parentheses are standard errors from 20 replicas.The underlying distribution is a product of (independent) piecewise constant densities with 10 discontinuities along each dimension. 63

4.11 Average CPU time in seconds of DET-CF(VC), DET-CF(EP), DET-CF(MLR), DET-CF(CM), and DET-CF(MaxEnt). The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a product of (independent) piecewise constant densities with 10 discontinuities along each dimension. 64

5.1 AUC scores for the IForest, LOF, DET(MLE) and KDE on various anomaly detection real datasets from the ODDS library. 74
5.2 Average Precision scores for the IForest, LOF, DET(MLE) and KDE on various anomaly detection real datasets from the ODDS library. 75
5.3 Classification rates for the Higgs, Letter, and MAGIC data: Random Forest, Kernel SVM, Logistic Regression, KDE and DET(MLE) 77

One

Introduction

Density estimation is an essential tool in statistics and sees applications in a variety of statistical and machine learning tasks. In exploratory analysis, density estimation is effective in presenting descriptive features of the underlying data distribution, such as modality, skewness and tail behavior. In clustering, one can find the level sets and modes of the density estimates, and samples belonging to the same component of the level sets or mode are associated with a cluster. Density estimation is also commonly applied as a preliminary step in regression, classification, goodness-of-fit testing, anomaly detection, and topological data analysis, etc.

As opposed to traditional parametric density estimation methods, nonparametric density estimation methods are considered more effective and flexible in various situations. A notable example of the superiority of the nonparametric model over a parametric one is provided in Park and Marron (1990), in which the authors study the distribution of net income data over a range of time. The density estimates from a lognormal model indicate unimodality and similar distribution from year to year, whereas a nonparametric model indicate bimodality and vast changes in distribution over time.

Formally, the problem of nonparametric density estimation is defined as follows.

Denote $(\mathbb{R}^d, \mu)$ as the d-dimensional Euclidean space with Lebesgue measure μ. Let $X_1, \dots, X_n \subset (\mathbb{R}^d, \mu)$ be n independent and identically distributed (*i.i.d.*) random samples generated from a probability distribution P with density function p, where p satisfies

$$p(x) \geq 0, \qquad \int p(x)\, d\mu(x) = 1. \tag{1.0.1}$$

The goal is to provide an estimate $\widehat{p}(x) = \widehat{p}(x; X_1, \dots, X_n)$ of p without assuming any parametric structure on the class of p. Note that even if we assume that the underlying measure space is the Euclidean space, the formulation in this thesis can be naturally extended to other measure spaces.

1.1 LITERATURE

There is a huge literature on nonparametric density estimation. In the following sections, we first review some of the classical density estimators in the literature. The drawbacks of the classical density estimators serve as the motivation for the research objective of this thesis: computationally efficient multivariate adaptive histograms. Next, we introduce some of the more recently developed methods in the literature.We give more details on the density estimators that are more relevant to the methods considered in this thesis (some of which are used as benchmarks in our numerical experiments), and we briefly mention the rest.

1.1.1 Classical Density Estimators

The Histogram

The histogram is probably the earliest density estimator introduced Karl (1895); yet, it is still considered one of the simplest and widely used density estimators nowadays. The method is often favored in exploratory analyses and in applications with large and high dimensional data because of its simplicity and computational efficiency. However, it's also well-known that the histogram does not have optimal convergence rate and suffers badly from the curse of dimensionality. The histogram estimator is formulated as the following.

Suppose that p is supported on the a hyper-cube $[0,1]^d$. Partition $[0,1]^d$ into a grid of equally sized bins, each with size h. Denote the bins as $B_1, \ldots, B_m$. The histogram estimator is defined as

$$\widehat{p}(x) = \frac{1}{h^d} \sum_{j=1}^{m} n_j \mathbb{I}(x \in B_j)$$

where

$$n_j = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(X_i \in B_j)$$

is the proportion of samples falling inside bin j.

Kernel Density Estimator

Kernel Density Estimation (KDE) is another classical density estimator and is formulated as follows. Given a kernel function $K(\cdot)$ and bandwidth $h = h_n > 0$,

$$\widehat{p}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K(\frac{\|x - X_i\|}{h}).$$

As opposed to the histogram, KDE achieves the minimax optimal rate of convergence with appropriately selected bandwidth. However, the vanilla KDE is computationally expensive, especially in high dimensions, because the method relies on computing pairwise distances between the observations. Moreover, in practice, the optimal bandwidth is unknown and the bandwidth selection process (e.g. cross-validation) can add to the runtime significantly. More efficient implementations of KDE can be achieved by sacrificing a bit of accuracy by utilizing tree-based data structures Gray and Moore (2003), Lee and Gray (2008), Beygelzimer et al. (2006) for the distance computations, or fast fourier transform (FFT) O'Brien et al. (2016) Gramacki and Gramacki (2017) for bandwidth selection.

1.1.2 Recent Advances

Another major issue with KDE and the histogram is that they do not adapt well to the heterogeneous smoothness of the data (i.e. the data can be smooth in some sections and wiggly in others), given that the bin size or bandwidth is taken to be a fixed value. In the multivariate case, both methods can be generalized by considering different values of bin size or bandwidth for each dimension (or more generally, a bandwidth matrix for KDE), but still, it does not solve of the problem of their inability to adapt to the heterogeneous smoothness within each dimension.

Naturally, with the objective of data adaptivity in mind, numerous extensions to the histogram and KDE have been developed. Adaptive KDE Terrell and Scott (1992) Shimazaki (2010), Wang and Wang (2007), k-nearest neighbor density estimator Mack and Rosenblatt (1979), and the rodeo KDE Liu et al. (2007) are some examples of data adaptive KDE methods. However, similar to the regular KDE, they also suffer from the same problem of computational inefficiency.

On the other hand, numerous efficient adaptive histogram-like methods have been developed recently. We say histogram-like because some of those estimators may not seem to be similarly formulated as the histogram; nonetheless, like the histogram, all of them provide piecewise-constant density estimates supported on axis-aligned hyper-rectangles. We introduce some of them here.

Essential histograms

The essential histogram Li et al. (2020) is a univariate piecewise-constant density estimator that is not necessarily designed to provide the best fit to the true density function, but rather serves the purpose of the histogram being a computationally efficient exploratory analysis tool. The method first constructs a confidence set of probability distributions that are able to estimate the density values as well as capturing the important features of the true density (e.g. modes, skewness), and then select a piecewise-constant distribution with the fewest number of bins in the confidence set. The authors show that the method is asymptotically optimal for detecting the important features. We give more a thorough introduction of the method in Section 3 and show how the method can be extended to higher dimensions.

Fused Density Estimator

The Fused Density Estimator (FDE) Bassett and Sharpnack (2019) is another univariate density estimator. The problem is formulated as the solution to a total variation penalized maximum likelihood problem. The authors show that the problem can be computed efficiently using quadratic programming, and the solution is a piecewise constant density function with discontinuities only occurring on the sample points. The estimator achieves the minimax rate of convergence in Hellinger distance over densities with log-bounded total variation. More formally, the problem is formulated as follows.

Given univariate samples $X_1, \dots, X_n \sim P$. Fix $\lambda > 0$, the FDE of p is the density $\widehat{p} = \exp(\widehat{g})$, where the log-density $\widehat{g}$ is the minimizer of the program

$$\widehat{g} \in \operatorname*{argmin}_{g \in \mathcal{G}} -\frac{1}{n}\sum_{i=1}^{n} g(X_i) + \lambda \mathrm{TV}(g) \quad \text{s.t.} \quad \int e^{g}\, d\mu(x) = 1$$

where $\mathcal{G} = \{g : \mathrm{TV}(g) < \infty\}$.

Tree-based Multivariate Histograms

In higher dimensions, multivariate histogram methods have been studied extensively in the past. Shang (1994), Sutton (1994), Ooi (2002), and Ram and Gray (2011) look at density estimation with CART Breiman et al. (1984)-type methods. The density estimation trees (DET) Ram and Gray (2011) method, in particular, is known to possess many desirable properties: it is consistent in L_2, highly interpretable, adapts well within each dimension and across dimensions, and has decent computational efficiency (good training time and extremely fast querying, which can be useful for predicting density values outside the sample points). The construction of DET is motivated by the widely used and successful classification and regression trees (CART) Breiman et al. (1984) in the supervised setting. The method proceeds by iteratively partitioning the space, and at each step, the optimal split is obtained by minimizing an estimate of the L_2 loss function on the densities. We will give a more detailed discussion of DET in Chapter 2 and propose a variation of it using the Kullback–Leibler (KL) divergence instead of the L_2 as the loss function.

Another class of tree-based density estimation methods in high dimension utilizes Bayesian approaches. Optional Polya Trees (OPT) Wong and Ma (2010) constructs a prior distribution using Polya trees with optional stopping and variable splitting rules. This prior distribution satisfies the Ferguson's criteria Ferguson (1973) and has a nice conjugate property in the sense that the posterior distribution is also an OPT. However, the exact computation of the OPT posterior in higher dimensions has high complexity and thus, in practice, one usually resorts to a more greedy approach (LL-OPT Jiang H (2016)) for faster computation. Bayesian Sequential Partitioning

(BSP) Lu et al. (2013) uses a different prior where the posterior distribution can be derived analytically. Inference from posterior is done via sequential importance sampling.

1.2 CONTRIBUTIONS

In this thesis, we propose several computational efficient multivariate adaptive histogram methods that are developed on top of the DET Ram and Gray (2011) estimator. In Chapter 2, we give a thorough introduction of DET: its mathematical formulation and algorithmic procedure. We propose a variant of DET by considering an alternative objective function, the KL divergence. In Chapter 3, in order to address some of the issues with DET estimators, we propose a class of methods for constructing efficient density estimators that naturally comes with confidence guarantees. The confidence sets we construct are finite sample, dimension free, and does not require any assumptions on the underlying distribution. Chapter 4 contains all the numerical experiments of the methods considered in Chapter 2 and 3. We give thorough numerical comparisons of our methods with classical density estimators in terms of estimation error in hellinger distance and CPU time. Our findings are backed by extensive low dimensional illustrations. In Chapter 5, we demonstrate the applicability of our methods by looking at problems in classification, density level set estimation, background subtraction, and anomaly detection. Finally, Chapter 6 provides an installation guide and a more detailed documentation to the package we developed for this thesis.

Two

Density Estimation Trees

Without loss of generality, assume that the samples $X_1, \ldots, X_n \in [0,1]^d$. Denote $\mathbf{\Pi}$ to be the class of partitions of $[0,1]^d$ using axis-aligned rectangles.

Definition 2.0.1. A **piecewise constant density function** p on partition $\Pi \in \mathbf{\Pi}$ is defined as:

$$p(x) = \sum_{\Omega_j \in \Pi} \beta_j \mathbb{I}(x \in \Omega_j), \quad x \in \mathbb{R}^d, \tag{2.0.1}$$

such that $\beta_j \geq 0$ and $\sum_j \beta_j \mu(\Omega_j) = 1$.

Denote $\mathcal{P}$ as the class of piecewise constant density functions. DET constructs an estimator $\widehat{p}$ of p by solving the optimization problem

$$\widehat{p} = \operatorname*{argmin}_{q \in \mathcal{P}} \widehat{L}(p, q), \tag{2.0.2}$$

where $\widehat{L}(p, \cdot)$ is an estimate of some loss function on p.

2.1 DET WITH L_2 LOSS

2.1.1 Problem Formulation

The original DET method proposed in Ram and Gray (2011) utilizes the L_2 loss as the loss function, i.e.

$$\begin{aligned} L_2(p,q) &= \int (p(x)-q(x))^2 \, d\mu(x) \\ &= \int p^2(x)\, dx - 2\int p(x)q(x)\, dx + \int q^2(x)\, dx. \end{aligned}$$

Note that the first term does not depend on q, the second term depends on the unknown density function p, but can be approximated using the monte carlo approximation $\int p(x)q(x)\,dx \approx \frac{1}{n}\sum_{i=1}^{n} q(X_i)$, and the last term is simply a function of q. Hence, we obtain an estimate of the L_2 loss and the DET estimator can be computed by solving

$$\widehat{p} = \operatorname*{argmin}_{q\in\mathcal{P}} -\frac{2}{n}\sum_{i=1}^{n} q(X_i) + \int q^2(x)\, dx. \tag{2.1.1}$$

Using the piecewise constant density formulation for q from definition (2.0.1), the objective function in (2.1.1) can be replaced with the following

$$-\frac{2}{n}\sum_{i=1}^{n}\sum_{\Omega_j\in\Pi} \beta_j \mathbb{I}(X_i\in\Omega_j) + \sum_{\Omega_j\in\Pi} \beta_j^2 \mu(\Omega_j). \tag{2.1.2}$$

For any fixed partition Π, by differentiating (2.1.2) with respect to each β_j and setting the derivative to zero, we get that

$$\beta_j = \frac{P_n(\Omega_j)}{\mu(\Omega_j)},$$

where $P_n(\Omega_j) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(X_i \in \Omega_j)$ is the fraction of samples falling inside hyper-rectangle Ω_j. Therefore, we can remove the dependence of problem (2.0.2) on the density values β_j and, after some simplifications, problem (2.0.2) boils down to constructing an optimal partition $\widehat{\Pi}$ of $[0,1]^d$ via

$$\widehat{\Pi} = \operatorname*{argmin}_{\Pi\in\mathbf{\Pi}} \widehat{R}(\Pi), \tag{2.1.3}$$

where

$$\widehat{R}(\Pi) = -\sum_{\Omega_j \in \Pi} \frac{P_n(\Omega_j)^2}{\mu(\Omega_j)}. \tag{2.1.4}$$

The optimal density estimator becomes

$$\widehat{p}(x) = \sum_{\Omega_j \in \widehat{\Pi}} \frac{P_n(\Omega_j)}{\mu(\Omega_j)} \mathbb{I}(x \in \Omega_j).$$

With (2.1.4) as the overall surrogate error, the greedy surrogate error for any hyperrectangle Ω can be likewise defined as

$$\widehat{R}(\Omega) = -\frac{P_n(\Omega)^2}{\mu(\Omega)}. \tag{2.1.5}$$

In practice, computing the argmin of $\widehat{R}(\Pi)$ is computationally intractable. The most common alternative is to greedily build a partition by splitting $[0,1]^d$ into smaller hyperrectangles recursively as in a tree-based structure.

2.1.2 Algorithm

DET utilizes the same procedure as CART Breiman et al. (1984) as its tree construction procedure. The tree growing, pruning and cross-validation steps are respectively modified to to account for the new loss function. The procedure takes into account the following model parameters

- `MAX_LEAF_SIZE`: a tree node will not get split on if the number of samples contained is less than `MAX_LEAF_SIZE`
- `MIN_LEAF_SIZE`: selection of the split location of each tree node ensures that there are at least `MIN_LEAF_SIZE` many samples in each of the two children tree nodes.

- `MTRY`: the number of variables available for splitting at each node. Note that this parameter was not included in the original DET implementation. We included this parameter because it is commonly used in constructing tree ensembles.
- `K_FOLD`: the number of folds used in the step of cross-validation for selecting the optimal pruning parameter α.

The DET algorithm takes the following three steps.

- Grow a density tree T using DET: Grow 1.
- Select the optimal pruning parameter α^* using DET: Cross-Validation 3.
- Prune T with pruning parameter α^* using DET: Prune 2.

Next, we discuss each of the steps in detail.

Growing. DET is constructed by recursively splitting each node into two children nodes. The optimal split location is selected as the one that maximally reduces (2.1.5). By convention, the candidate split location is taken to be the the middle points of adjacent samples in each dimension. A node does not get split on if the number of samples contained is less than `MAX_LEAF_SIZE` and we select the split location that guarantees at least `MIN_LEAF_SIZE` many samples in each of its children nodes. The complete algorithm is given in Algorithm 1.

Pruning. Once the tree T is fully grown, in order to prevent overfitting, the tree is pruned using the minimal cost-complexity pruning procedure introduced in Breiman et al. (1984). Denote the subtree of T rooted at a node Ω as T_Ω, and the subtree omitting T_Ω as $T - T_\Omega$. We abuse notations here and denote the loss of a tree

Algorithm 1 DET: Grow

Inputs: Ω – current tree node, X – the set of samples in the current node

```
if $|X| \le$ MAX_LEAF_SIZE then
    return $Node(Val \leftarrow \frac{|X|}{N\mu(\Omega)})$
else
    Randomly sample (without replacement) MTRY many covariates from $1, \dots, d$ and denote the subset as $V$.
    Let $S = \{\}$ be the set of candidate split locations. Denote $X^j_{(i)}$ as the value of the $i^{th}$ largest sample in coordinate $j$.
    For each $v \in V$, add $(X^v_{(i)} + X^v_{(i+1)})/2$ into $S$ for all $i =$ MIN_LEAF_SIZE, $\dots$, $n-$MIN_LEAF_SIZE.
    Find the optimal split $s^*$ of the current node $\Omega$ into $\Omega^{s^*}_L$ and $\Omega^{s^*}_R$ such that
    $s^* = \text{argmax}_{s \in S} \widehat{R}(\Omega) - \widehat{R}(\Omega^s_L) - \widehat{R}(\Omega^s_R)$.
    return $Node(Left \leftarrow grow(X^{s^*}_L, \Omega^{s^*}_L), Right \leftarrow grow(X^{s^*}_R, \Omega^{s^*}_R))$
end if
```

T as $\widehat{R}(T) = \sum_{\Omega:\text{leaves in } T} \widehat{R}(\Omega)$, the sum of the loss over all its leave nodes.The penalized surrogate error of T with pruning parameter α is defined as $\widehat{R}_\alpha(T) = \widehat{R}(T) + \alpha|T|$, where $|T|$ is the number of leaves in T. Therefore, a node Ω is pruned if $\widehat{R}_\alpha(T - T_\Omega) - \widehat{R}_\alpha(T) \le 0$, or equivalently, $\alpha \le \frac{\widehat{R}(\Omega) - \widehat{R}(T_\Omega)}{|T_\Omega| - 1}$. Since the tree constructed in the previous step has finitely many nodes, there are finitely many values for the fraction. The algorithm looks for the nodes to be pruned by sequentially increasing the fraction values, until it is bigger than the given α value. The complete algorithm is given in Algorithm 2.

Cross-Validation. Finally, the optimal α in the pruning step is selected using leave-one-out cross validation (LOO-CV) or K-fold cross-validation (KCV). The complete algorithm with K-fold cross-validation is given in Algorithm 3.

Algorithm 2 DET: Prune

```
Inputs: $T$ – a decision tree, $\alpha$ – regularization parameter
  Initialize $T^1 = T$, $\alpha_1 = 0$, $i = 1$.
  while $\alpha_i \leq \alpha$ do
    For a given node $\Omega$ in $T^i$, define $g_i(\Omega) = \frac{\widehat{R}(\Omega) - \widehat{R}(T^i_\Omega)}{|T^i_\Omega| - 1}$.
    Select tree node $\Omega_i$ in $T^i$ by $\Omega_i = \operatorname{argmin}_{\Omega \in T_i} g_i(\Omega)$.
    Let $\alpha_{i+1} = g_i(\Omega_i)$ and $T^{i+1} = T^i - T^i_{\Omega_i}$.
    $i = i + 1$
  end while
  return $T^i$
```

Algorithm 3 DET: Cross-Validation

```
Inputs: $T$ – a decision tree, $K$ – K_FOLD
  Split data into $K$ folds, denote the $X^{(i)}$ as the data in the $i^{th}$ fold, and $X^{-(i)}$ as the data in all but the $i^{th}$ fold.
  Define $J_i(\alpha) = \int (\widehat{p}_i^\alpha(x))^2 \, dx + \frac{2}{|X^{-(i)}|} \sum_{z \in X^{-(i)}} \widehat{p}_i^\alpha(z)$ for $i = 1, \ldots, K$, where $\widehat{p}_i^\alpha$ is the estimator built on $X^{(i)}$ with pruning parameter $\alpha$.
  Define $J(\alpha) = \frac{1}{K} \sum_{i=1}^K J_i(\alpha)$.
  Optimal regularization parameter is the solution to $\alpha^* = \operatorname{argmin}_\alpha J(\alpha)$.
  return $\alpha^*$
```

2.1.3 Computational Complexity

Table 2.1 summarizes the training and querying computational complexity of KDE, tree-based KDE, OPT, LL-OPT, and DET algorithms. The training complexity, $\mathcal{O}(Hdn^2)$, of a vanilla fixed bandwidth KDE comes from selecting the optimal bandwidth parameter from a candidate set of H many choices. Given an estimated bandwidth, the computational complexity of computing the density estimate of a single query for KDE is $\mathcal{O}(dn)$. Spatial partitioning using the tree-based data structure allows more efficient computation of the KDE Gray and Moore (2003), Lee and Gray (2008). In particular, with cover trees Beygelzimer et al. (2006), the training time complexity can be improved to $\mathcal{O}(Hdn)$ Ram et al. (2009) and the querying time to $\mathcal{O}(d \log n)$.

Methods	Training	Querying
Histogram	$\mathcal{O}(Hdn)$	$\mathcal{O}(d)$
KDE	$\mathcal{O}(Hdn^2)$	$\mathcal{O}(dn)$
Tree-based KDE	$\mathcal{O}(Hdn)$	$\mathcal{O}(d\log n)$
OPT	$\mathcal{O}(nd^{n/c})$	$\mathcal{O}(D_T)$
LL-OPT	$\mathcal{O}(n(n+2^h d)d^h)$	$\mathcal{O}(D_T)$
DET	Slow KCV	$\mathcal{O}(D_T)$

Table 2.1: Training and Querying computational complexity of KDE, tree-based KDE, OPT, LL-OPT, and DET.

In addition, the querying time for $\mathcal{O}(n)$ many queries can be amortized, yielding an order of $\mathcal{O}(dn)$ total time Ram et al. (2009). The training and querying complexity of the histogram are $\mathcal{O}(Hdn)$ and $\mathcal{O}(d)$, respectively. The training time comes from the selection of the optimal bin size parameter, with H being the number of candidate choices.

The training computational complexity of OPT Wong and Ma (2010) and LL-OPT Jiang H (2016) are $\mathcal{O}(nd^{n/c})$ and $\mathcal{O}(n(n+2^h d)d^h)$, respectively, where h and c are some pre-determined model parameters. The detailed complexity derivations for both methods are provided in Jiang H (2016). The analytical expression for the computational complexity of the full training stage of DET is unknown, due to the intricacies in the pruning and cross-validation step. Generally, the complexity for growing a decision tree scales like $\mathcal{O}(dn\log n)$ Hastie et al. (2001), and the computational bottleneck of the DET algorithm comes from the cross-validation step. DET, OPT, and LL-OPT all enjoy extremely efficient querying time due to their tree-based structures; the complexity is of order $\mathcal{O}(D_T)$, where D_T is the depth of the resulting decision tree. D_T has a worst case upper bound of $\mathcal{O}(n)$, but is in practice seen to be close to $\mathcal{O}(\log n)$.

2.2 DET with Maximum Likelihood

2.2.1 Problem Formulation

While being a common and simple loss function, the L_2 loss may not be a good loss for density estimation as it tends to downweight low density regions. Alternatively, we propose a variant of DET by considering the KL divergence as the loss function,

$$\begin{aligned} L_{\mathrm{KL}}(p, q) &= \int \log \frac{p(x)}{q(x)} p(x)\, d\mu(x) \\ &= \int \log p(x) p(x)\, d\mu(x) - \int \log q(x) p(x)\, d\mu(x). \end{aligned}$$

Similar to the derivations in the previous section, after expanding the expression for the KL divergence, we may drop the first term since it's independent of q, and approximate the second term using monte carlo approximations. We arrive at an estimate of the KL divergence and the DET estimator can be computed by solving

$$\widehat{p} = \underset{q \in \mathcal{P}}{\operatorname{argmin}} -\frac{1}{n} \sum_{i=1}^{n} \log q(X_i). \tag{2.2.1}$$

Note that minimizing the KL divergence is equivalent to maximizing the likelihood, we refer to this DET method as DET(MLE) and the original one as DET(L_2).

After plugging in expression for piecewise constant density function from definition (2.0.1), the objective function in (2.2.1) can be replaced with the following

$$\begin{aligned} & -\frac{1}{n} \sum_{i=1}^{n} \sum_{j} \log(\beta_j) \mathbb{I}(X_i \in \Omega_j) \\ = & -\sum_{j} \log(\beta_j) P_n(\Omega_j) \end{aligned}$$

For a fixed partition Π, β_j can be computed exactly by solving a simple constrained maximization problem,

$$\max_{\beta_j} \sum_j \log(\beta_j) P_n(\Omega_j) \quad \text{subject to} \quad \sum_j \beta_j \mu(\Omega_j) = 1.$$

The Lagragian dual (with dual variable λ) to the program is given as the following

$$\min_{\lambda} \max_{\beta_j} \sum_j \log(\beta_j) P_n(\Omega_j) - \lambda(\sum_j \beta_j \mu(\Omega_j) - 1).$$

By differentiating the objective function with respect to β_j and setting the derivative to 0, we obtain that

$$\frac{1}{\beta_j} P_n(\Omega_j) - \lambda \mu(\Omega_j) = 0 \implies \beta_j = \frac{P_n(\Omega_j)}{\lambda \mu(\Omega_j)}$$

Thus,

$$\sum_j \beta_j \mu(\Omega_j) = 1 \implies \sum_j \frac{P_n(\Omega_j)}{\lambda \mu(\Omega_j)} \mu(\Omega_j) = 1 \implies \lambda = 1,$$

which implies that $\beta_j = P_n(\Omega_j)/\mu(\Omega_j)$, and problem (2.0.2) boils down to constructing an optimal partition $\widehat{\Pi}$ of $[0,1]^d$ via

$$\widehat{\Pi} = \operatorname*{argmin}_{\Pi} \widehat{R}(\Pi),$$

where

$$\widehat{R}(\Pi) = -\sum_{\Omega_j \in \Pi} P_n(\Omega_j) \log \frac{P_n(\Omega_j)}{\mu(\Omega_j)}. \tag{2.2.2}$$

The optimal density estimator becomes

$$\widehat{p}(x) = \sum_{\Omega_j \in \widehat{\Pi}} \frac{P_n(\Omega_j)}{\mu(\Omega_j)} \mathbb{I}(x \in \Omega_j).$$

With (2.2.2) as the overall surrogate error, the greedy surrogate error for any hyper-rectangle Ω can be likewise defined as

$$\widehat{R}(\Omega) = -P_n(\Omega) \log \frac{P_n(\Omega)}{\mu(\Omega)}. \tag{2.2.3}$$

2.2.2 Algorithm

The procedure for DET(MLE) is identical to that of DET(L_2), except for a different choice of the greedy surrogate loss function. We ask the readers to refer to Section 2.1.2 for the complete algorithm.

2.2.3 Computational Complexity

The computational complexity of DET(MLE) is identical to that of DET(L_2). See Section 2.1.3 for more details.

2.3 BAGGED DET

Booststrap Aggregating, or Bagging, is a common technique applied to decision trees in regression and classification settings for variance reduction. In Chapter 4, we investigate the effectiveness of bagging on DET. The procedure of a Bagged DET is given in Algorithm 4.

Algorithm 4 Bagged DET

```
Inputs: $B$ – number of bags, $m$ – MTRY
  for $b = 1, \ldots, B$ do
    Draw a bootstrap sample $\mathbb{Z}$ of size $n$ from the observed sample $X_1, \ldots, X_n$.
    Construct a DET on $\mathbb{Z}$ using the algorithms described in Section 2.1.2 with MTRY $= m$. Denote the estimator as $\widehat{p}^b$.
  end for
  Let $\widehat{p}(x) = \frac{1}{B} \sum_{b=1}^{B} \widehat{p}^b(x)$.
  return $\widehat{p}$
```

2.4 SUMMARY

In summary, in this chapter, we provided a thorough introduction of the DET estimator proposed by Ram and Gray (2011). We described in detail how the DET estimator is mathematically formulated, followed by its algorithmic description and computational complexity analysis. Next, we propose a variant of the DET method, which we refer to as DET(MLE), by considering the KL divergence as the loss function. The algorithm for constructing DET(MLE) is almost identical to that of original DET method; both utilize a CART-like procedure, which consists of a decision tree growing step, a pruning step, and a cross-validation step. Finally, we describe how bagging can be applied to the DET estimators.

Three

Density Estimation with Confidence

3.1 DET with Confidence

The DET estimators have some potential limitations. First, when we have a large amount of samples, the DET estimators can occasionally overfit, and thereby producing a number of local ”spikes”, or spurious nodes in the density estimates (see Figure 4.1 as an example). The spikes does not necessarily affect the estimation errors, since they usually have tiny probability mass, but may be misleading in downstream analyses. For example, the spikes can show up as an extra cluster in cluster analyses, or simply, give an inaccurate estimates of the number of modes. Note that in all our experiments, we are already setting the regularization parameters `MAX_LEAF_SIZE` $= 10$ and `MIN_LEAF_SIZE` $= 5$. The reason for this behavior comes from the greedy pruning process.

The second limitation of the DET estimators is its lack of uncertainty quantification. While a single density estimator gives the best estimator according to its model, it does not naturally come with a level of confidence. Generally, confidence statements can be provided for simpler density estimators under certain smoothness assumptions on p. For example, confidence intervals or confidence bands can be calculated with the histogram and KDE because both are well understood estimators with complete risk analysis results; see Scott (2015), Wasserman (2006), Tsybakov (2008), Giné and Guillou (2002), Einmahl and Mason (2005). However, for more complicated estimators such as the DET, it is not as straightforward how those confidence statements can be constructed. In addition, obtaining confidence bands from confidence intervals generally involves computationally expensive procedures such as the bootstrap Bickel and Rosenblatt (1973), Rosenblatt (1976), Neumann (1998), Chernozhukov et al. (2016), Chernozhukov et al. (2013), which is computationally impractical on large high-dimensional datasets.

Finally, as explained in Section 2.1.3, the computational bottleneck of the DET algorithm comes from the cross-validation step for variable selection of the pruning parameter, and thereby limiting the computational scalability of the method to large datasets.

With these limitations in mind, we propose an alternative way of constructing an density estimator that naturally comes with confidence guarantees. The idea is to first construct a confidence set, and then select an estimator inside the confidence set. The essential histogram Li et al. (2020) is one such example in the univariate case. The construction of the confidence set utilizes sample splitting and level sets of the density function. Sample splitting is a powerful tool that sees use in a variety of statistical inference problems. For example, in conformal predictive inference Shafer and Vovk (2008), Lei et al. (2016), Vovk and Buntine (2012), sample splitting is a key element in the construction of valid prediction sets in the regression and classification setting. Notably, conformal methods do not require any assumptions on the underlying density or model, and the prediction sets are valid in finite samples. Sample splitting is also used in hypothesis testing. The universal inference approach developed by Wasserman et al. (2020) provides a new likelihood ratio testing framework that addresses situations where the classical approach is not valid. Sample splitting allows the construction of a test and confidence set that are valid in finite samples and without regularity conditions. The motivation for using upper level sets as part of the confidence set has several reasons. First, probability density functions can be (almost surely) uniquely defined by upper level sets, i.e. for probability distributions P and Q with density functions p and q, respectively, $P(A) = Q(A)$ for all upper level sets A of p and q if and only if $P = Q$ almost surely. In addition, Polonik (1999) argues that upper level sets are natural extensions of quantiles in goodness-of-fit tests. The corresponding test statistics, similar to the univariate case, behaves asymptotically as the uniform empirical process under appropriate assumptions.

In the following sections, we discuss three ways to construct such confidence sets. The first two methods harness the explicit form of upper level sets and the third one is an extension of the essential histogram approach to higher dimensions, which also indirectly uses probability statements of the level sets. The three confidence sets we construct are all valid in finite sample, dimension free, and does not require any smoothness assumptions of the underlying density.

The problem setup is as follows. Given *i.i.d.* random samples $X_1, \dots, X_n \sim P$ on $\mathbb{R}^d$, we randomly split $\{1, \dots, n\}$ into two subsets $\mathcal{I}_1$ and $\mathcal{I}_2$ with size n_1 and n_2, respectively. Generally, $\{X_i : i \in \mathcal{I}_1\}$ is referred as the training subset, used for fitting the model of interest, and $\{X_i : i \in \mathcal{I}_2\}$ is referred as the validation set, used for constructing the confidence bands. Let $\widetilde{p}$ be any density estimator constructed on the training set, we denote the upper level set of $\widetilde{p}$ as $\mathcal{L}$, i.e.

$$\mathcal{L} = \{L_t : t \in \mathbb{R}\} \text{ where } L_t = \{x : \widetilde{p}(x) > t\}.$$

Let $\widetilde{P}_n$ be the empirical probability measure with respect to the validation set.

3.1.1 VC-based Confidence Set

The first method relies on the following result from Vapnik-Chervonenkis (VC) theory. This result is a direct consequence of the relative VC inequality (see Theorem 7 from Bousquet et al. (2004)) and the Sauer's lemma.

Theorem 3.1.1. Chaudhuri et al. (2014) Let $\mathcal{F}$ be a class of functions from $\mathbb{R}^d$ to $\{0,1\}$ with VC dimension $h < \infty$, and P a probability distribution on $\mathbb{R}^d$. Suppose n samples are drawn independently at random from P; let P_n denote the empirical distribution with respect to this sample. Then for any $\delta > 0$, with probability at least $1-\delta$, the following holds for all $f \in \mathcal{F}$,

$$-\min\left(\beta_\delta\sqrt{P_n f}, \beta_\delta^2 + \beta_\delta\sqrt{Pf}\right) \le Pf - P_n f \le \min\left(\beta_\delta^2 + \beta_\delta\sqrt{P_n f}, \beta_\delta\sqrt{Pf}\right),$$

where $\beta_\delta = \sqrt{(4/n)(h\log(2en/h) + \log(4/\delta))}$.

Next, note that the class $\mathcal{L}$ has VC dimension 1, because level sets are embedded in one another. After some rearrangements, we get that for any $\delta > 0$, with probability at least $1-\delta$, for any $L_t \in \mathcal{L}$,

$$P(L_t) \in \Bigg[\max\left(0, \widetilde{P}_n(L_t) - \beta_\delta\sqrt{\widetilde{P}_n(L_t)}, \widetilde{P}_n(L_t) - \beta_\delta^2/2 - \beta_\delta\sqrt{\widetilde{P}_n(L_t) - 3\beta_\delta^2/4}\right), \tag{3.1.1}$$

$$\min\left(1, \widetilde{P}_n(L_t) + \beta_\delta^2/2 + \beta_\delta\sqrt{\widetilde{P}_n(L_t) + \beta_\delta^2/4}\right)\Bigg].$$

where $\beta_\delta = \sqrt{(8/n)(\log(en) + \log(4/\delta))}$.

Finally, let the set $C_P(t,\delta)$ be the set of probability distributions that satisfies (3.1.1). Note that the confidence statement holds uniformly for all $L_t \in \mathcal{L}$. In practice, we take an arbitrarily fine grid of $t = t_1, \ldots, t_K$ to produce a $1-\delta$ confidence set for

P.

$$\mathbb{P}(P \in \bigcap_{i=1}^{K} C_P(t_i, \delta)) \geq 1 - \delta. \tag{3.1.2}$$

The confidence set derived from VC inequalities is a simple one, but is conservative because the constants in Theorem 3.1.1 are not sharp. Next, we describe alternative procedures for deriving sharper confidence sets.

3.1.2 Empirical Process Based Confidence Set

Define random variable $Y = \widetilde{p}(X)$ and $Y_i := \widetilde{p}(X_i)$ for $i \in \mathcal{I}_2$. Denote the distribution function of Y as F_Y, and the empirical distribution function w.r.t. $\{Y_i : i \in \mathcal{I}_2\}$ as $\widetilde{F}_{n,Y}$. Then, for any $L_t \in \mathcal{L}$, we have

$$P(L_t^c) = P(\{x : \widetilde{p}(x) \leq t\}) = F_Y(t)$$

$$\widetilde{P}_n(L_t^c) = \frac{1}{n_2} \sum_{i \in \mathcal{I}_2} \mathbb{I}(\widetilde{p}(X_i) \leq t) = \widetilde{F}_{n,Y}(t)$$

Consider the statistic

$$Z := \sup_{t \in \mathbb{R}} \frac{\sqrt{n_2}[P(L_t) - \widetilde{P}_n(L_t)]}{\sqrt{P(L_t)(1 - P(L_t))}} = \sup_{t \in \mathbb{R}} \frac{\sqrt{n_2}[\widetilde{F}_{n,Y}(t) - F_Y(t)]}{\sqrt{F_Y(t)(1 - F_Y(t))}}$$

If F_Y is continuous, the distribution of Z does not depend on F_Y, we may substitute F_Y and $\widetilde{F}_{n,Y}$ with the distribution function of a uniform random variable. In other words, let $U_1, \ldots, U_{n_2} \overset{i.i.d}{\sim} \text{Uniform}[0,1]$,

$$Z \overset{d}{=} \sup_{0 \leq t \leq 1} \frac{\sqrt{n_2}}{\sqrt{t(1-t)}} \left[\frac{1}{n_2} \sum_{i=1}^{n_2} \mathbb{I}(U_j \leq t) - t \right]$$

Note that even if F_Y is not continuous, the confidence set we derive assuming F_Y is continuous is still valid, yet conservative. Let the order statistics of $U_1, \ldots, U_{n_2}$ be

$U_{(1)}, \ldots, U_{(n_2)}$, then

$$\frac{\sqrt{n_2}}{\sqrt{t(1-t)}}\left[\frac{1}{n_2}\sum_{i=1}^{n_2}\mathbb{I}(U_j \le t) - t\right] = \begin{cases} -\frac{\sqrt{n_2}}{\sqrt{t(1-t)}}t, & \text{for } 0 \le t < U_{(1)} \\ \vdots & \\ \frac{\sqrt{n_2}}{\sqrt{t(1-t)}}\left[\frac{k-1}{n_2} - t\right], & \text{for } U_{(k-1)} \le t < U_{(k)} \\ \vdots & \\ \frac{\sqrt{n_2}}{\sqrt{t(1-t)}}(1-t), & \text{for } U_{(n_2)} \le t \le 1 \end{cases}$$

Hence, setting $U_{(0)} = 0$, $U_{(n_2+1)} = 1$ and using the convention $0/0 = 0$, we have

$$\begin{aligned} \sup_{0\le t\le 1}\frac{\sqrt{n_2}}{\sqrt{t(1-t)}}\left[\frac{1}{n_2}\sum_{i=1}^{n_2}\mathbb{I}(U_j \le t) - t\right] &= \sup_{1\le k\le n_2+1}\frac{\sqrt{n_2}}{\sqrt{U_{(k-1)}(1-U_{(k-1)})}}\left[\frac{k-1}{n_2} - U_{(k-1)}\right] \\ &= \sup_{0\le k\le n_2}\frac{\sqrt{n_2}}{\sqrt{U_{(k)}(1-U_{(k)})}}\left[\frac{k}{n_2} - U_{(k)}\right] \end{aligned} \tag{3.1.3}$$

since $(a-t)/\sqrt{t(1-t)}$ is a decreasing function in t for $a \in [0,1]$. Similarly, consider the statistic

$$Z' := \sup_{t\in\mathbb{R}} -\frac{\sqrt{n_2}[P(L_t) - \widetilde{P}_n(L_t)]}{\sqrt{P(L_t)(1-P(L_t))}} = \sup_{t\in\mathbb{R}} -\frac{\sqrt{n_2}[\widetilde{F}_{n,Y}(t) - F_Y(t)]}{\sqrt{F_Y(t)(1-F_Y(t))}}$$

Then,

$$\begin{aligned} Z' &\overset{d}{=} \sup_{0\le t\le 1}\frac{\sqrt{n_2}}{\sqrt{t(1-t)}}\left[t - \frac{1}{n_2}\sum_{i=1}^{n_2}\mathbb{I}(U_j \le t)\right] \\ &= \sup_{1\le k\le n_2+1}\frac{\sqrt{n_2}}{\sqrt{U_{(k)}(1-U_{(k)})}}\left[U_{(k)} - \frac{k-1}{n_2}\right]. \end{aligned} \tag{3.1.4}$$

Next, we generate B many times the random variable in (3.1.3) and denote them as $Z_1 \ldots, Z_B$, and generate B many times the random variable in (3.1.4) and denote them as $Z'_1, \ldots, Z'_B$. Let

$$M := \max(Z, Z') \quad \text{and} \quad M_b = \max\left(Z_b, Z'_b\right) \text{ for } b = 1, \ldots, B.$$

Define

$$\xi_{\alpha,\delta} = \sup_z \left\{ z : \sum_{b=1}^{B} \mathbb{I}(M_b \leq z) \leq F^{-1}_{\mathrm{Bin}(B,1-\alpha)}(1-\delta) \right\},$$

where we denote $F_{\mathrm{Bin}(n,p)}(\cdot)$ as the distribution function of a $\mathrm{Bin}(n,p)$ random variable.

Lemma 3.1.2.

$$\mathbb{P}(\mathbb{P}(\max(Z, Z') \leq \xi_{\alpha,\delta} | Z_1, \ldots, Z_B, Z'_1, \ldots, Z'_B) \geq 1-\alpha) \geq 1-\delta.$$

Proof.

$$\begin{aligned}\mathbb{P}(\xi_{\alpha,\delta} < c) &= \mathbb{P}(\sum_{b=1}^{B} \mathbb{I}(M_b \leq c) > F^{-1}_{\mathrm{Bin}(B,1-\alpha)}(1-\delta)) \\ &= 1 - F_{\mathrm{Bin}(B,F_M(c))}(F^{-1}_{\mathrm{Bin}(B,1-\alpha)}(1-\delta)).\end{aligned}$$

Thus,

$$\begin{aligned} & \mathbb{P}(\mathbb{P}(\max(Z, Z') \leq \xi_{\alpha,\delta} | Z_1, \ldots, Z_B, Z'_1, \ldots, Z'_B) \geq 1-\alpha) \\ = \ & \mathbb{P}(F_M(\xi_{\alpha,\delta}) \geq 1-\alpha) \\ = \ & \mathbb{P}(\xi_{\alpha,\delta} \geq F_M^{-1}(1-\alpha)) \\ = \ & F_{\mathrm{Bin}(B,F_M(F_Z^{-1}(1-\alpha)))}(F^{-1}_{\mathrm{Bin}(B,1-\alpha)}(1-\delta)) \\ \geq \ & F_{\mathrm{Bin}(B,1-\alpha)}(F^{-1}_{\mathrm{Bin}(B,1-\alpha)}(1-\delta)) \\ \geq \ & 1-\delta \end{aligned}$$

□

This implies that with probability at least $1-\delta$, the generated data $Z_1, \ldots, Z_B, Z'_1, \ldots, Z'_B$ leads to a $1-\alpha$ quantile of Z and Z'. Namely, with probability at least $1-\alpha$,

$$\sup_{t\in\mathbb{R}} \frac{\sqrt{n_2}[P(L_t) - \widetilde{P}_n(L_t)]}{\sqrt{P(L_t)(1-P(L_t))}} \leq \xi_{\alpha,\delta} \quad \text{and} \quad \sup_{t\in\mathbb{R}} -\frac{\sqrt{n_2}[P(L_t) - \widetilde{P}_n(L_t)]}{\sqrt{P(L_t)(1-P(L_t))}} \leq \xi_{\alpha,\delta}.$$

After some rearrangements, we have

$$P(L_t) \in \left[\frac{\widetilde{P}_n(L_t) + \frac{\xi^2_{\alpha,\delta}}{2n_2} \pm \sqrt{\frac{\xi^4_{\alpha,\delta}}{4n_2^2} + \frac{\widetilde{P}_n(L_t)(1-\widetilde{P}_n(L_t))\xi^2_{\alpha,\delta}}{n_2}}}{1+\xi^2_{\alpha,\delta}/n_2} \right] \text{ for all } \ t\in\mathbb{R}. \tag{3.1.5}$$

Denote $C_P(t,\delta)$ as the set of probability distributions that satisfies (3.1.5). Note that the confidence statement holds uniformly for all $L_t \in \mathcal{L}$. In practice, we can take an arbitrarily fine grid of $t = t_1, \ldots, t_K$ to produce a $1-\delta$ confidence set for P.

$$\mathbb{P}(P \in \bigcap_{i=1}^{K} C_P(t_i,\delta)) \geq 1-\delta. \tag{3.1.6}$$

3.1.3 Multiscale Likelihood Ratio Tests

In Section 1, we briefly introduced the essential histogram, a univariate adaptive histogram method. Here, we show how sample splitting allows us to extend the framework of the essential histogram to higher dimensions.

More formally, the (univariate) essential histogram is constructed as follows. Given univariate samples $Z_1, \ldots, Z_m \sim P$, one can invert the multiscale likelihood ratio test to construct a $1-\delta$ confidence set for P.

$$\left\{ Q : (2\log \mathrm{LR}_m(Q(I), P_m(I)))^{1/2} \leq \ell(P_m(I)) + \kappa_m(\delta) \text{ for all } I \in \mathcal{J} \right\}, \tag{3.1.7}$$

where P_m is the empirical distribution on $Z_1, \ldots, Z_m$,

$$\log \mathrm{LR}_m(Q(I), P_m(I)) = mP_m(I)\log\left(\frac{P_m(I)}{Q(I)}\right) + m(1-P_m(I))\log\left(\frac{1-P_m(I)}{(1-Q(I))}\right)$$

is the log-likelihood statistic for testing $P(I) = Q(I)$,

$$\ell(P_m(I)) = \left(2\log\left(\frac{e}{P_m(I)(1-P_m(I))}\right)\right)^{1/2}$$

is the scale penalty, and $\kappa_m(\delta)$ is the $1-\delta$ quantile of the distribution of

$$T_m = \max_{I\in\mathcal{J}}\left\{(2\log \mathrm{LR}_m(Q(I), P_m(I)))^{1/2} - \ell(P_m(I))\right\}.$$

$\mathcal{J}$ is a collection of intervals,

$$\mathcal{J} = \bigcup_{l=2}^{l_{\max}} \mathcal{J}(I), \quad \text{where } \; l_{\max} = \left\lfloor \log_2 \frac{m}{\log m}\right\rfloor,$$

$$\mathcal{J}(l) = \left\{(Z_{(j)}, Z_{(k)}] : j,k \in \{1 + id_l, i \in \mathbb{N}_0\} \cap \mathcal{D}, m_l < k - j \leq 2m_l\right\},$$
$$\text{where } \; m_l = m2^{-l}, d_l = \left\lceil \frac{m_l}{6l^{1/2}} \right\rceil, \text{ and } \mathcal{D} = \{i : Z_{(i)} \neq Z_{(i+1)}\}.$$

The essential histogram estimator is the probability distribution in (3.1.7) with the fewest number of bins and can be computed efficiently using dynamic programming.

Note that the framework for the essential histogram can be extended to higher dimensions if we consider carrying out the multiscale likelihood ratio test on the transformed variables $Y_i = \widetilde{p}(X_i)$ for $i \in \mathcal{I}_2$. For any interval $I \subset \mathbb{R}$, let $\widetilde{p}^{-1}(I) = \{x : \widetilde{p}(x) \in I\}$ be the pre-image of I under $\widetilde{p}$. We have

$$\frac{1}{n_2}\sum_{i\in\mathcal{I}_2} \mathbb{I}(Y_i \in I) = \widetilde{P}_n(\widetilde{p}^{-1}(I)) \qquad \text{and} \qquad \mathbb{P}(Y \in I) = \mathbb{P}(X \in \widetilde{p}^{-1}(I)).$$

Define

$$C_P(\delta) := \left\{Q : \left(2\log \mathrm{LR}_{n_2}(Q(\widetilde{p}^{-1}(I)), \widetilde{P}_n(\widetilde{p}^{-1}(I)))\right)^{1/2} \leq \ell(\widetilde{P}_n(\widetilde{p}^{-1}(I))) + \kappa_{n_2}(\alpha) \text{ for all } I \in \mathcal{J}\right\}, \tag{3.1.8}$$

Therefore, $C_P(\delta)$ is a $1-\delta$ confidence set for P, i.e.,

$$\mathbb{P}(P \in C_P(\delta)) \geq 1 - \delta.$$

3.1.4 Algorithm

Having introduced the confidence sets, next we discuss how we can construct a DET-like estimator that belongs to the confidence set. The idea is simple. We grow the density tree iteratively one node at a time, until the density tree belongs to the confidence set of interest. However, unlike the classical DET method, the tree nodes are not split in a recursive depth-first order. Instead, the tree is split iteratively using a priority queue data structure that stores the set of all leaf nodes, where the priority score of each leaf node is its maximum reduction in the surrogate error if split. At each iteration, we first check if the confidence statements are satisfied. If satisfied, we return the current tree, and if not, we select the node in the priority queue with the highest score, and split the node using Algorithm 5. The optimal split dimension and value are selected using Algorithm 6. Once the selected node is split, its two children nodes are added into the priority queue. The complete algorithm is given in Algorithm 7.

Algorithm 5 SplitNode

Inputs: *node* – current node, Ω – hyper-rectangle of current node, X – the set of samples in the current node

```
if $|X| >$ MAX_LEAF_SIZE then
    $s^* \leftarrow node.splitLoc$
    $node.left \leftarrow node(X_L^{s^*}, \Omega_L^{s^*}), node.right \leftarrow node(X_R^{s^*}, \Omega_R^{s^*})$
    FindSplit($node.left$).
    FindSplit($node.right$).
end if
```

Choice of $\widetilde{p}$. Notice that all three confidence methods involves computing the probability mass of $\widetilde{p}^{-1}([a,b])$, for some $0 \leq a \leq b \leq \infty$. For piecewise constant density estimators, $\widetilde{p}^{-1}([a,b])$ is simply a union of disjoint hyperrectangles. In addition, for

Algorithm 6 FindSplit

Inputs: *node* – current node, Ω – hyper-rectangle of current node, X – the set of samples in the current node

```
Randomly sample (without replacement) MTRY many variables and denote the subset as $V$.
Let $S = \{\}$ be the set of candiate split locations. Denote $X_i^j$ as the value of $j^{\text{th}}$ coordinate of sample $X_i$.
For each variable $v \in V$, add $(X_i^v + X_{i+1}^v)/2$ into $S$ for all $i =$ MIN_LEAF_SIZE, $\ldots$, $n-$ MIN_LEAF_SIZE.
Find the optimal split $s^*$ of the current node $\Omega$ into $\Omega_L^{s^*}$ and $\Omega_R^{s*}$ such that $s^* = \operatorname{argmax}_{s \in S} R(\Omega) - R(\Omega_L^s) - R(\Omega_R^s)$.
$node.split \leftarrow s^*$.
$node.loss \leftarrow R(\Omega) - R(\Omega_L^{s^*}) - R(\Omega_R^{s^*})$.
```

Algorithm 7 DET-CF

Inputs: $X_1, \ldots, X_n$ – samples

```
Construct initial density estimator $\widetilde{p}$ on the training set$\{X_i\}_{i \in \mathcal{I}_1}$.
Construct a confidence set $C$ using $\widetilde{p}$ and the validation set $\{X_i\}_{i \in \mathcal{I}_2}$ using one of (3.1.2), (3.1.6), and (3.1.8).
Let $pq$ = PriorityQueue() where the priority scores are determined by the object's surrogate loss reduction ($object.loss$).
$root = node(X_1, \ldots, X_n)$.
FindSplit($root$).
while $root \notin C$ do
    $node = pq.pop()$.
    SplitNode($node$).
    if $node.left.loss > 0$ then
        $pq.push(node.left)$.
    end if
    if $node.right.loss > 0$ then
        $pq.push(node.right)$.
    end if
end while
return $root$
```

tree based density estimators such as the DET and OPT, the probability mass of the hyperrectangles can be queried very efficiently, and thereby making them the ideal choices for the initial density estimator $\widetilde{p}$. In all our experiments in Chapter 4, we will be using DET(MLE) as the initial density estimator for all DET with confidence methods.

Speed-up. Note that the corresponding confidence sets of all three confidence statements (3.1.2), (3.1.6), and (3.1.8) can be written as the intersection of larger confidence sets: for the VC and empirical process based confidence sets, the intersection is taken over the upper level sets of the $\widetilde{p}$, and for the multiscale likelihood ratio test based confidence set, the intersection is taken over the collection of intervals. Therefore, checking the validity of each three confidence statements is equivalent to checking a sequence of simpler confidence statements. However, such a procedure can be quite computationally costly if we have a large number of statements to check. One caveat we can make is that when we are splitting a node deeper down in the density tree, the hyperrectangle associated may only be relevant to a small part of the list of confidence statements, because the rest of the confidence statements do not depend on the probability content of the current node. Therefore, by keeping track of the list of relevant confidence statements as the nodes are split, we can greatly reduce the number of confidence statements to check as the tree grows bigger.

3.2 Optimization Methods

Although the greedy algorithm 7 provides a very efficient way of constructing a DET-like density estimator inside the confidence set of interest, a successful construction is actually not guaranteed. There are situations where a density tree is fully grown while parts of the list of confidence statements remain unsatisfied. Intuitively, the density tree is grown by greedily splitting the nodes that yield the maximum reduction in the surrogate loss, and thus, generally, high probability density regions that contains more samples get split on much more often than low probability regions. As a result, certain confidence statements involving low density regions may remain unsatisfied throughout the entire process. This behavior can happen more regularly in higher dimensional datasets than lower dimensional ones; due to the curse of dimensionality, the volume of the space increases so fast that the available data become sparse.

Next, we discuss two approaches that guarantee finding a density estimator inside the confidence set of interest. Both methods are framed as constrained convex optimization problems, and thus can be solved using appropriate convex solvers. The solutions to both methods still return piecewise constant density estimates; however, the training and querying time are significantly higher than the greedy DET-CF method 7.

Recall that checking the whether a probability measure Q satisfies any of the three confidence statements (3.1.2), (3.1.6), and (3.1.8) is equivalent to checking a list of simpler confidence statements of the form $L_j \leq Q(\widetilde{p}^{-1}(I_j)) \leq U_j$, $j = 1, \ldots, K$ for some $K > 0$. For the VC and empirical process based confidence methods, I_j is simply an open interval of the form $[a, \infty)$ for some $a > 0$, and L_j and U_j are the corresponding lower and upper bounds in the confidence intervals (3.1.1) and (3.1.5), respectively. For the multiscale likelihood ratio test based confidence set, I_j comes from the collection of intervals in $\mathcal{J}$, and L_j and U_j can be computed using root searching algorithms.

3.2.1 Constrained MLE

In the first approach, the density estimator is constructed as the solution to a constrained MLE problem. We restrict our attention to the class of piecewise constant densities with identical support as $\widetilde{p}$. More formally, let Π be the partition associated with $\widetilde{p}$, the problem can be formulated as

$$\begin{aligned} &\widehat{p} = \operatorname*{argmax}_{g \in \mathcal{G}} \sum_{i=1}^{n} \log g(X_i) \\ &\text{subject to } \; L_j \leq \int_{\widetilde{p}^{-1}(I_j)} g(x)\, d\mu(x) \leq U_j \; \text{ for all } j = 1, \ldots, K \end{aligned} \tag{3.2.1}$$

where $\mathcal{G} = \{g : g(x) = \sum_i a_i \mathbb{I}(x \in \Omega_i), a_i \geq 0, \int g(x)\, d\mu(x) = 1\}$. The problem can be equivalently formulated in matrix form. Since Π is fixed, the only free variables are the density values a_i's. Denote $\boldsymbol{a} = (a_1, \ldots, a_{|\Pi|})$, $\boldsymbol{v} = (\mu(\Omega_1), \ldots, \mu(\Omega_{|\Pi|}))$, $\boldsymbol{L} = (L_j)_{j \in [K]}$, $\boldsymbol{U} = (U_j)_{j \in [K]}$. Let $\boldsymbol{M}$ be a $K \times |\Pi|$ matrix where $M_{ij} = \mu(\Omega_j)\mathbb{I}(\Omega_j \in \widetilde{p}^{-1}(I_i))$ and $\boldsymbol{H}$ be a $n \times |\Pi|$ matrix where $H_{ij} = \mathbb{I}(X_i \in \Omega_j)$. Problem (3.2.1) can be equivalently formulated as

$$\begin{aligned} &\widehat{\boldsymbol{a}} = \operatorname*{argmax}_{\boldsymbol{a}} \log(\boldsymbol{a})^T \boldsymbol{H}\boldsymbol{1} \\ &\text{subject to } \; \boldsymbol{L} \leq \boldsymbol{M}\boldsymbol{a} \leq \boldsymbol{U}, \boldsymbol{a}^T \boldsymbol{v} = 1, \boldsymbol{a} \geq \boldsymbol{0} \end{aligned}$$

The resulting density estimator is

$$\widehat{p}(x) = \sum_i \widehat{a}_i \mathbb{I}(x \in \Omega_i).$$

3.2.2 Constrained Maximum Entropy

In the second approach, the density estimator is taken as the solution to a constrained maximum entropy problem (MaxEnt). Compared to the previous approach, this formulation does not put any constraint on the class of the underlying density function, yet we can still obtain a piecewise constant density estimator. The MaxEnt approach for density estimation was first proposed in Jaynes (1957), and later sees applications in various areas, such as statistical mechanics Kapur and Kesavan (1992), and natural language processing Berger et al. (1996), Della Pietra et al. (1997).In MaxEnt, one is given a set of constraints on the target distribution P; usually the constraints require the expectations of a certain set of feature functions (real-valued functions) with respect to the target distribution match their empirical means. Using the Karush-Kuhn-Tucker (KKT) optimality conditions, one can show that the problem can be equivalently formulated as a maximum likelihood estimation problem, where the target distribution belongs to the class of *Gibbs distributions* Dudík (2011).

In Dudík et al. (2004), the authors discuss several limitations of the original MaxEnt formulation, and propose using a relaxation of the feature-based constraints; the new constraints only require the the feature expectation to be within a certain range of their empirical means. The authors show that this new formulation is equivalent to solving a maximum likelihood problem with a ℓ_1 style regularization term, with the candidate distribution also belonging to the class of *Gibbs distributions*. Next, we give a more formal introduction to the MaxEnt problem, and demonstrate how it can be applied to finding a density estimator satisfying any of the three confidence statements (3.1.2), (3.1.6), and (3.1.8).

The entropy of a distribution P with density function p is defined as

$$H(p) = -\int p(x) \log p(x) \, d\mu(x).$$

Let $f_j(x) = \mathbb{I}(x \in \widetilde{p}^{-1}(I_j))$ and $\boldsymbol{f} = (f_j)_{j \in [K]}$ be the feature functions. Consider all

Gibbs distribution of the form

$$q_{\boldsymbol{\lambda}}(x) = \frac{e^{\boldsymbol{\lambda}\cdot \boldsymbol{f}(x)}}{Z_{\boldsymbol{\lambda}}},$$

where $Z_{\boldsymbol{\lambda}} = \int e^{\boldsymbol{\lambda}\cdot \boldsymbol{p}(x)}\, d\mu(x)$ is a normalizing constant and $\boldsymbol{\lambda} \in \mathbb{R}^d$. The MaxEnt problem is formulated as follows.

$$\begin{aligned} &\max_g H(g) \\ \text{s.t.} &\int g(x)\, d\mu(x) = 1, \\ \text{and } L_j \leq &\int_{\widetilde{p}^{-1}(I_j)} g(x)\, d\mu(x) \leq U_j \quad \text{for all } j = 1, \ldots, K. \end{aligned} \tag{3.2.2}$$

For simplicity, denote $G_j := \int_{\widetilde{p}^{-1}(I_j)} g(x)\, d\mu(x)$. We take a similar procedure as in Dudík et al. (2004) to derive the Lagrangian dual of (3.2.2). The dual problem (with dual variables $\lambda_0, \lambda_j^+, \lambda_j^-$) can be written as

$$\min_{\lambda_j^+, \lambda_j^-, \lambda_0} \max_g H(g) - \lambda_0\Big(\int g(x)\, d\mu(x) - 1\Big) - \sum_I \lambda_j^+ (L_j - G_j) - \sum_I \lambda_j^- (G_j - U_j).$$

After rearranging the terms, we arrive at

$$\min_{\lambda_j^+, \lambda_j^-, \lambda_0} \max_g H(g) - \lambda_0\Big(\int g(x)\, d\mu(x) - 1\Big) - \sum_I (\lambda_j^+ - \lambda_j^-) G_j + \sum_I \lambda_j^- U_j + \sum_I \lambda_j^+ L_j.$$

Taking the derivative with respect to $g(x)$ and setting it to zero yields that $g(x)$ must be a Gibbs distribution with parameters $\boldsymbol{\lambda} = (\lambda_j)_{j\in[K]}$ and $\lambda_0 + 1 = \log Z_{\boldsymbol{\lambda}}$, where $\lambda_j = \lambda_j^+ - \lambda_j^-$. Let $\boldsymbol{U} = (U_j)_{j\in[K]}$ and $\boldsymbol{L} = (L_j)_{j\in[K]}$, problem (3.2.2) can be equivalently formulated as

$$\widehat{\boldsymbol{\lambda}}_j^+, \widehat{\boldsymbol{\lambda}}_j^- = \operatorname*{argmin}_{\boldsymbol{\lambda}_j^+, \boldsymbol{\lambda}_j^-} H(q_{\boldsymbol{\lambda}}) + \boldsymbol{\lambda} \cdot q_{\boldsymbol{\lambda}}(g) + \boldsymbol{\lambda}^- \cdot \boldsymbol{U} - \boldsymbol{\lambda}^+ \cdot \boldsymbol{L},$$

which can be simplified into

$$\widehat{\boldsymbol{\lambda}}_j^+, \widehat{\boldsymbol{\lambda}}_j^- = \operatorname*{argmin}_{\boldsymbol{\lambda}_j^+, \boldsymbol{\lambda}_j^-} \log Z_{\boldsymbol{\lambda}} + \boldsymbol{\lambda}^- \cdot \boldsymbol{U} - \boldsymbol{\lambda}^+ \cdot \boldsymbol{L}. \tag{3.2.3}$$

The resulting density estimator is simply

$$\widehat{p}(x) = q_{\widehat{\boldsymbol{\lambda}}}(x),$$

where $\widehat{\boldsymbol{\lambda}} = \widehat{\boldsymbol{\lambda}}_j^+ - \widehat{\boldsymbol{\lambda}}_j^-$. Note that $\widehat{p}$ is a piecewise constant density function because the feature functions f_j are indicator functions in this case.

Note that (3.2.3) is an unconstrained convex program, which can be solved using existing convex solvers. However, when we have a large set of constraints, second order optimization methods may impose too much memory pressure. Alternatively, one can sacrifice some accuracy and runtime efficiency for better memory efficiency using first order sequential-update algorithms, such as the coordinate descent. The procedure described below is adapted from Dudík et al. (2004). For each j, define $\widetilde{\lambda}_j^+ = \lambda_j^+ + \delta^+$ and $\widetilde{\lambda}_j^- = \lambda_j^- + \delta^-$, where $\delta^+, \delta^- \in \mathbb{R}$. Let $L(\cdot)$ be the objective function in (3.2.3), then

$$\begin{aligned} L(\widetilde{\boldsymbol{\lambda}}^+, \boldsymbol{\lambda}^-) - L(\boldsymbol{\lambda}^+, \boldsymbol{\lambda}^-) &= \log Z_{\widetilde{\boldsymbol{\lambda}}^+ - \boldsymbol{\lambda}^-} - \log Z_{\boldsymbol{\lambda}} - L_j(\widetilde{\lambda}_j^+ - \lambda_j^+) \\ &= \log q_{\boldsymbol{\lambda}}[e^{\delta^+ f_j}] - L_j \delta^+ \\ &\le \log(1 + (e^{\delta^+} - 1) q_{\boldsymbol{\lambda}}[f_j]) - L_j \delta^+ \\ &=: F_j^+(\boldsymbol{\lambda}^+, \boldsymbol{\lambda}^-, \delta). \end{aligned} \tag{3.2.4}$$

Similarly,

$$\begin{aligned} L(\boldsymbol{\lambda}^+, \widetilde{\boldsymbol{\lambda}}^-) - L(\boldsymbol{\lambda}^+, \boldsymbol{\lambda}^-) &= \log Z_{\boldsymbol{\lambda}^+ - \widetilde{\boldsymbol{\lambda}}^-} - \log Z_{\boldsymbol{\lambda}} + U_j(\widetilde{\lambda}_j^- - \lambda^-) \\ &= \log q_{\boldsymbol{\lambda}}[e^{-\delta^- f_j}] + U_j \delta^- \\ &\le \log(1 + (e^{-\delta^-} - 1) q_{\boldsymbol{\lambda}}[f_j]) + U_j \delta^- \\ &=: F_j^-(\boldsymbol{\lambda}^+, \boldsymbol{\lambda}^-, \delta). \end{aligned} \tag{3.2.5}$$

The coordinate descent algorithm for solving problem 3.2.3 is given in Algorithm 8. At each step, the algorithm modifies one λ_j^+ or λ_j^- at a time. The algorithm can be proved to be convergent by using a similar argument as in Dudík et al. (2004).

Algorithm 8 MaxEnt: coordinate-descent

```
Inputs: T – number of iterations.
  Let $\boldsymbol{\lambda}_1^+ = 0, \boldsymbol{\lambda}_1^- = 0$.
  for $t = 1, \ldots, T$ do
    Let $(j, \delta^+) = \mathrm{argmin}_{j,\delta} F_j^+(\boldsymbol{\lambda}_t^+, \boldsymbol{\lambda}_t^-, \delta)$ and $(j, \delta^-) = \mathrm{argmin}_{j,\delta} F_j^-(\boldsymbol{\lambda}_t^+, \boldsymbol{\lambda}_t^-, \delta)$,
    where $F_j^+(\cdot)$ and $F_j^-(\cdot)$ are defined in (3.2.4) and (3.2.5).
    if $F_j^+(\boldsymbol{\lambda}_t^+, \boldsymbol{\lambda}_t^-, \delta^+) \leq F_j^-(\boldsymbol{\lambda}_t^+, \boldsymbol{\lambda}_t^-, \delta^-)$ then
      $\lambda_{t+1,j'}^+ = \begin{cases} \lambda_{t,j}^+ + \delta^+ & \text{if } j' = j \\ \lambda_{t,j'}^+ & \text{else} \end{cases}$
    else
      $\lambda_{t+1,j'}^- = \begin{cases} \lambda_{t,j}^- + \delta^- & \text{if } j' = j \\ \lambda_{t,j'}^- & \text{else} \end{cases}$
    end if
  end for
  return $\boldsymbol{\lambda}_T^+, \boldsymbol{\lambda}_T^-$
```

3.3 SUMMARY

In summary, in this chapter, we discuss some of the potential limitations of the DET estimators, and propose another class of DET-like methods that are intended to address these issues. More specifically, we propose three ways for constructing confidence sets of the underlying density, and provide a greedy algorithm for efficiently finding a DET-like density estimator that belongs to the confidence set. The confidence sets we constructed are dimension free, finite sample, and does not require any regularity assumption of the underlying distribution. Next, we address the potential issue of the greedy algorithm not being able to converge to a density estimator in the confidence set. The idea is to alternatively formulate problem as constrained convex optimization problems. The first method constructs a density estimator by solving a constrained MLE problem. The second method constructs the density estimator using ideas from maximum entropy density estimation.

Four

Numerical Experiments

4.1 DET(L_2) AND DET(MLE)

In this section, we demonstrate the performance of DET estimators. To show the efficiency and scalability of DET methods, we first compare them with KDE and OPT in terms of estimation error and running time. Next we provide some concrete lower dimensional illustrations of the DET estimators in comparison with other state-of-the-art density estimators mentioned in the Introduction 1. Finally, we show that the DET estimators can be utilized as a variable selection tool.

Both DET methods are implemented in `C++` with bindings to Python and R. Please refer to Chapter 6 for more implementation details. Unless otherwise specified, for all our experiments, we are using the default model parameters, i.e. `MAX_LEAF_SIZE` = 10, `MIN_LEAF_SIZE` = 5, `MTRY` = D, and the number of folds $K = 10$. For KDE, we are using the `KernelDensity()` function from `sklearn` in `python` with default models parameters except for the bandwidth. In the hellinger distance and running time comparison experiements, we set the bandwidth using Silverman's rule of thumb for simplicity. For the lower dimensional experiments, the bandwidth is selected using 5-fold cross-validation. Note that the `KernelDensity()` utilizes k-d tree and ball tree for more efficient computation of the nearest neighbors. For OPT, we are using the

LL-OPT implementation Jiang H (2016) with two look ahead levels ($h = 2$). All experiments are simulated on a Macbook Pro with Apple M1 Pro chip with 16GB of RAM.

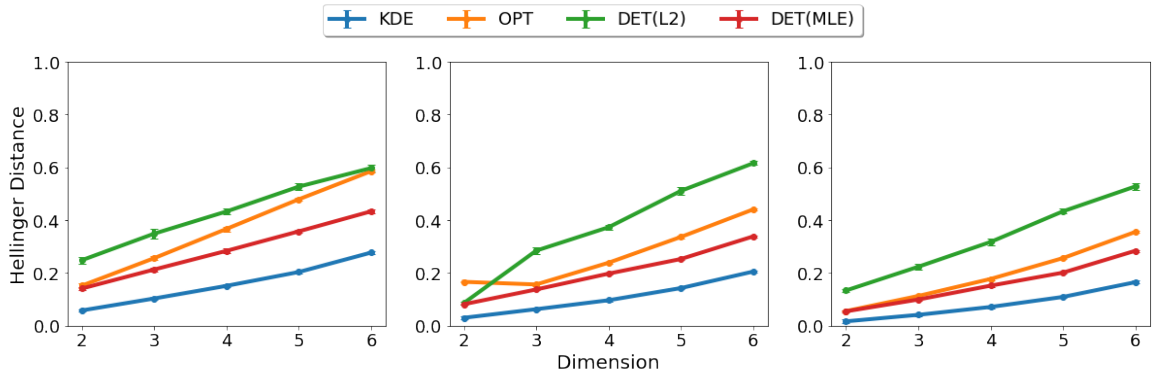
4.1.1 Accuracy and Speed

First, we compare the performance of DET(L_2) and DET(MLE) with that of KDE with a fixed bandwidth and OPT in terms of Hellinger distance and running time.

Mixture of Gaussians

We simulate samples $X \sim \sum_{i=1}^{4} \frac{1}{4}\mathcal{N}(\mu_i, \Sigma_i)$ with $D = 2, 3, 4, 5, 6$ and $N = 2000, 20000, 100000$ respectively. The mean vectors and covariance matrices for each component are generated randomly for each (N, D) pair. As a preprocessing step, we standardize each covariate such that they have mean zero and standard deviation one, which is necessary for fitting KDE with a fixed bandwidth. This processing step can be omitted for the DET methods as they do not have any model parameters depending on the scale of the data.

Table 4.1 records the average hellinger distance between the true density and the estimation from each method over 20 replications. Table 4.2 records the average CPU time of each method in seconds over the 20 replications. The numbers in the parenthesis are the corresponding standard deviation. In general, KDE gives the best estimation error, followed by DET(MLE) and OPT. Note the performance of KDE can still be significantly improved for optimally selected bandwidth (using cross-validation) or with adaptive (bandwidth) KDE methods at higher computational cost. DET(MLE) and OPT gives similar performance in lower dimensions, and DET(MLE) improves over OPT by a fair margin as the dimension gets higher. DET(L_2), on the other hand, is a bit lacking in performance,especially in higher dimensions.



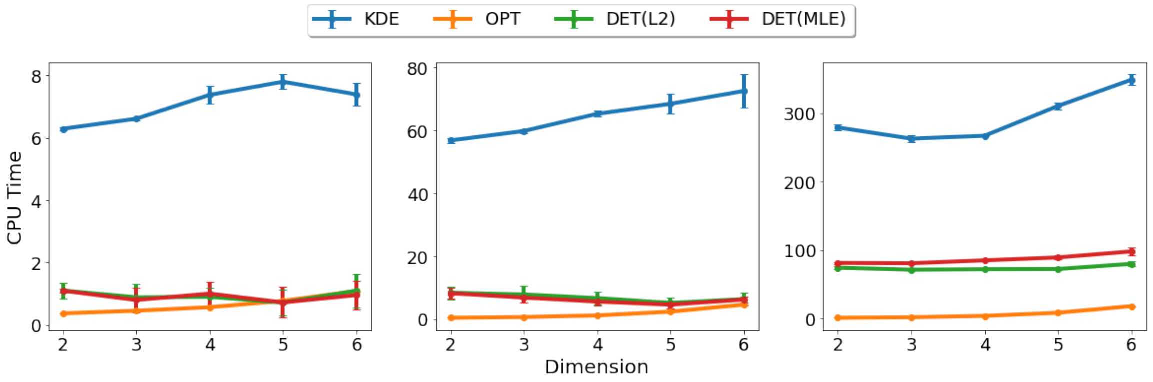
	$N = 2000$				$N = 20000$				$N = 100000$			
D	KDE	OPT	DET(L_2)	DET(MLE)	KDE	OPT	DET(L_2)	DET(MLE)	KDE	OPT	DET(L_2)	DET(MLE)
2	0.0583	0.1520	0.2483	0.1411	0.0305	0.1666	0.0862	0.0818	0.0170	0.0559	0.1337	0.0549
	(0.0053)	(0.0085)	(0.0139)	(0.0065)	(0.0050)	(0.0028)	(0.0061)	(0.0024)	(0.0069)	(0.0027)	(0.0067)	(0.0038)
3	0.1035	0.2561	0.3489	0.2132	0.0631	0.1566	0.2846	0.1377	0.0417	0.1131	0.2238	0.0996
	(0.0064)	(0.0083)	(0.0180)	(0.0085)	(0.0032)	(0.0049)	(0.0135)	(0.0040)	(0.0041)	(0.0032)	(0.0112)	(0.0031)
4	0.1513	0.3669	0.4329	0.2838	0.0969	0.2389	0.3732	0.1977	0.0716	0.1776	0.3177	0.1521
	(0.0048)	(0.0093)	(0.0113)	(0.0086)	(0.0046)	(0.0050)	(0.0100)	(0.0026)	(0.0057)	(0.0044)	(0.0125)	(0.0046)
5	0.2038	0.4791	0.5270	0.3577	0.1429	0.3366	0.5110	0.2530	0.1095	0.2565	0.4336	0.2015
	(0.0044)	(0.0055)	(0.0116)	(0.0051)	(0.0044)	(0.0049)	(0.0147)	(0.0043)	(0.0035)	(0.0034)	(0.0093)	(0.0037)
6	0.2774	0.5855	0.5973	0.4335	0.2056	0.4408	0.6157	0.3387	0.1653	0.3556	0.5278	0.2842
	(0.0051)	(0.0065)	(0.0134)	(0.0063)	(0.0038)	(0.0053)	(0.0072)	(0.0044)	(0.0052)	(0.0045)	(0.0134)	(0.0037)

Table 4.1: Error in Hellinger Distance between the true density and KDE, OPT, DET(L_2), and DET(MLE). The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a mixture of four Gaussian distributions.

In terms of computational efficiency, OPT is the fastest of all. Both DET methods are significantly faster KDE, especially for large samples sizes and higher dimensions, which proves a big advantage of DET estimators over KDE in higher dimensions. DET(MLE), while enjoying much better Hellinger error distance, is only slightly slower than DET(L_2). It is also worth emphasizing that the DET methods have extremely efficient querying time ($\sim 10^{-4}$ secs), which can be particularly useful for predicting density values outside the samples points.

Piecewise Constant Densities

Similar to the Gaussian scenario, Table 4.3 and Table 4.4 give the estimation error and running time when the data are generated from piecewise constant densities. The



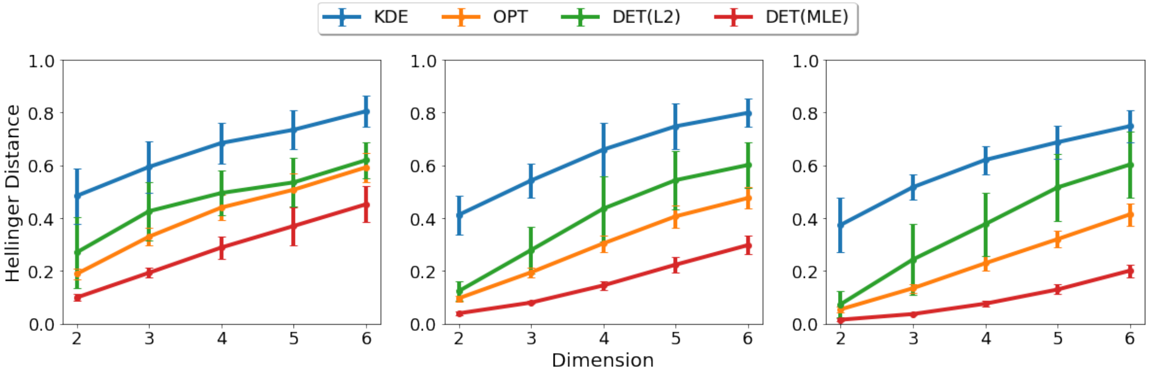
	$N = 2000$				$N = 20000$				$N = 100000$			
D	KDE	OPT	DET(L_2)	DET(MLE)	KDE	OPT	DET(L_2)	DET(MLE)	KDE	OPT	DET(L_2)	DET(MLE)
2	6.2923	0.3701	1.1010	1.0960	56.8166	0.4489	8.3994	8.2408	278.9758	0.8157	73.9431	80.9630
	(0.0411)	(0.0073)	(0.2473)	(0.0668)	(0.8027)	(0.0034)	(2.0411)	(1.6330)	(3.9961)	(0.0051)	(1.7722)	(1.7984)
3	6.6136	0.4539	0.8832	0.7967	59.8095	0.6804	7.8331	6.8500	262.6469	1.6202	71.0551	80.4940
	(0.0439)	(0.0039)	(0.4383)	(0.3874)	(0.5662)	(0.0036)	(2.6684)	(1.4359)	(4.5769)	(0.0081)	(0.9687)	(0.5833)
4	7.3774	0.5669	0.8998	1.0016	65.2782	1.1865	6.6886	5.5885	266.8435	3.5486	71.7238	84.7116
	(0.2877)	(0.0057)	(0.3058)	(0.3714)	(0.9712)	(0.0119)	(2.1540)	(1.2716)	(1.1531)	(0.0312)	(0.9816)	(0.8013)
5	7.7992	0.7705	0.7130	0.7194	68.4494	2.3876	5.1550	4.5996	310.3625	8.1626	72.0560	88.9800
	(0.2353)	(0.0049)	(0.4084)	(0.4911)	(3.1933)	(0.0428)	(1.6922)	(1.0634)	(5.5689)	(0.0657)	(1.1836)	(2.8023)
6	7.3934	1.0849	1.0963	0.9541	72.5337	4.6280	6.3683	6.2312	348.6712	17.7740	79.5784	97.7803
	(0.3696)	(0.0162)	(0.5338)	(0.4436)	(5.2640)	(0.0531)	(1.9639)	(1.0442)	(7.7749)	(0.1978)	(3.4399)	(5.8268)

Table 4.2: Average CPU time in seconds of KDE, OPT, DET(L_2), and DET(MLE). The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a mixture of four Gaussian distributions.

density function is generated as a product of D 1-dimensional piecewise constant density function, where the number of discontinuities along each dimension is 10. We observe similar behaviors among the four estimators as in the Gaussian case, except that the Hellinger distance for both versions of DET are now much lower than that of KDE, which is expected as DET is designed to provide piecewise constant estimates. Note that in this case we also have DET(MLE) outperform DET(L_2) and OPT in Hellinger distance.

4.1.2 Adaptability

Next, to better understand the behaviours of the DET methods, we look at some concrete examples in lower dimensions.



	$N = 2000$				$N = 20000$				$N = 100000$			
D	KDE	OPT	DET(L_2)	DET(MLE)	KDE	OPT	DET(L_2)	DET(MLE)	KDE	OPT	DET(L_2)	DET(MLE)
2	0.4843	0.1890	0.2702	0.0996	0.4124	0.0966	0.1236	0.0398	0.3744	0.0548	0.0740	0.0157
	(0.1049)	(0.0216)	(0.1341)	(0.0125)	(0.0731)	(0.0109)	(0.0358)	(0.0064)	(0.1018)	(0.0106)	(0.0505)	(0.0046)
3	0.5941	0.3300	0.4264	0.1934	0.5424	0.1948	0.2791	0.0804	0.5174	0.1348	0.2437	0.0372
	(0.0981)	(0.0326)	(0.1093)	(0.0181)	(0.0636)	(0.0195)	(0.0876)	(0.0054)	(0.0484)	(0.0169)	(0.1341)	(0.0047)
4	0.6846	0.4403	0.4953	0.2890	0.6596	0.3044	0.4361	0.1454	0.6205	0.2294	0.3774	0.0764
	(0.0777)	(0.0482)	(0.0836)	(0.0420)	(0.1008)	(0.0315)	(0.1237)	(0.0162)	(0.0532)	(0.0273)	(0.1203)	(0.0114)
5	0.7347	0.5078	0.5355	0.3704	0.7478	0.4069	0.5440	0.2240	0.6874	0.3207	0.5164	0.1300
	(0.0731)	(0.0631)	(0.0924)	(0.0714)	(0.0870)	(0.0416)	(0.1113)	(0.0281)	(0.0621)	(0.0312)	(0.1260)	(0.0183)
6	0.8047	0.5925	0.6196	0.4527	0.7989	0.4763	0.6013	0.2981	0.7484	0.4144	0.6030	0.2014
	(0.0603)	(0.0561)	(0.0682)	(0.0673)	(0.0541)	(0.0410)	(0.0870)	(0.0343)	(0.0607)	(0.0422)	(0.1262)	(0.0240)

Table 4.3: Error in Hellinger Distance between the true density and KDE, OPT, DET(L_2), and DET(MLE). The numbers in parentheses are standard errors from 20 replicas.The underlying distribution is a product of (independent) piecewise constant densities with 10 *discontinuities along each dimension.*

1D examples. We first compare the performance of DET(MLE), DET(L_2), OPT, and KDE with two other state-of-the-art density estimators in one dimension — essential histograms and FDE. Figure 4.1, 4.2, and 4.3 give the case when the underlying densities are generated from the claw distribution (mixture of 5 Gaussian distributions with unequal mean and same variance), harp distribution (mixture of 5 Gaussian distributions with unequal mean and variance), and a piecewise constant distribution with 10 discontinuities respectively.

While KDE is unable to adapt well to the heterogeneous degree of smoothness of the densities in our examples, all the other five estimators are doing decent work. In general, we notice that both DET estimators give very similar performance to that of

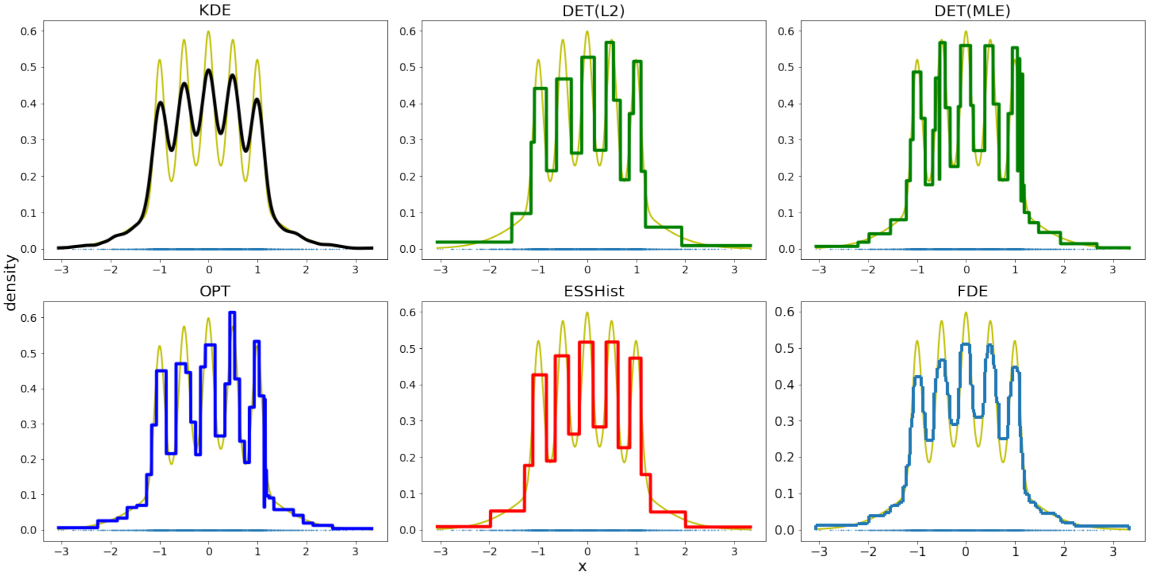


Figure 4.1: Density Estimates for KDE, DET(L_2),DET(MLE), OPT,Essential Histogram, and FDE when the underlying distribution is a univariate claw distribution. Sample size is 5000.

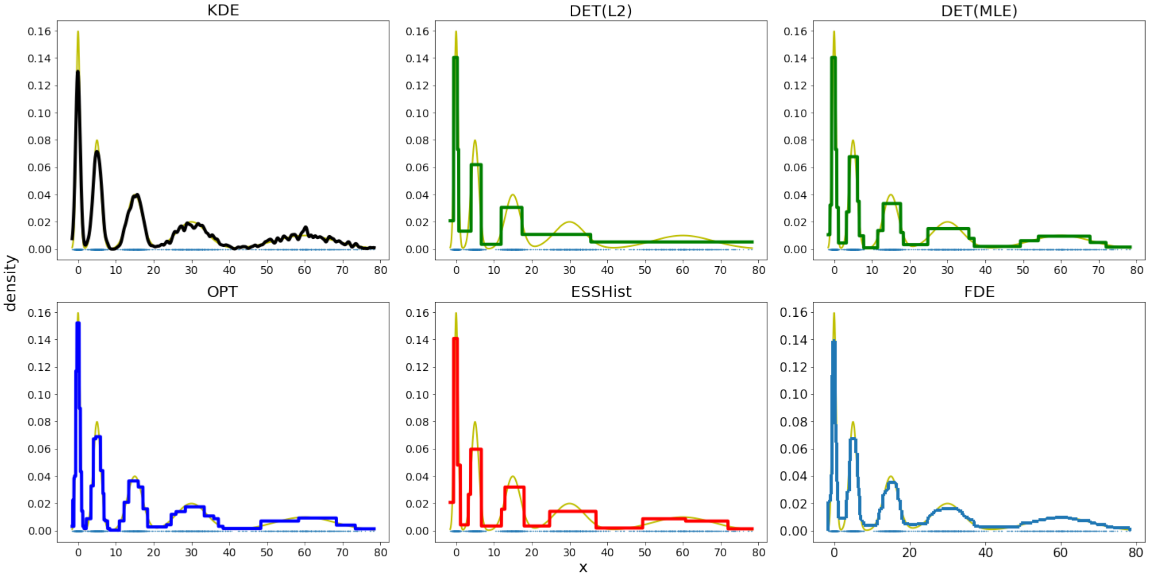
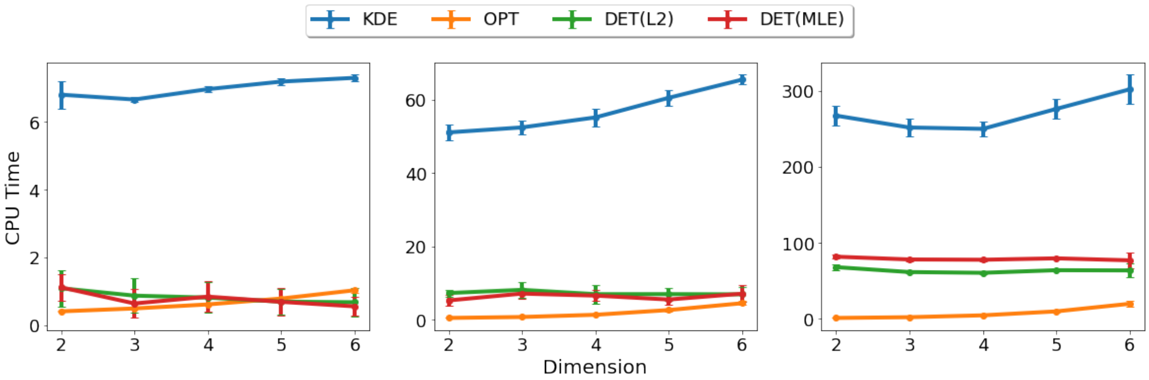


Figure 4.2: Density Estimates for KDE, DET(L_2),DET(MLE), OPT,Essential Histogram, and FDE when the underlying distribution is a univariate harp distribution. Sample size is 5000.



	$N = 2000$				$N = 20000$				$N = 100000$			
D	KDE	OPT	DET(L_2)	DET(MLE)	KDE	OPT	DET(L_2)	DET(MLE)	KDE	OPT	DET(L_2)	DET(MLE)
2	6.7972	0.4064	1.0874	1.1135	51.0509	0.5005	7.2771	5.2695	267.2345	0.8658	67.7634	81.5085
	(0.4111)	(0.0095)	(0.5432)	(0.3842)	(2.0758)	(0.0119)	(0.9354)	(1.4265)	(13.4730)	(0.0301)	(3.9599)	(2.2978)
3	6.6550	0.4920	0.8697	0.6387	52.4089	0.7573	8.1489	7.1278	251.6021	1.8684	61.2846	77.8866
	(0.0724)	(0.0093)	(0.5045)	(0.4276)	(1.9858)	(0.0430)	(2.1772)	(1.4061)	(12.1612)	(0.1496)	(1.9087)	(2.5618)
4	6.9609	0.6110	0.8114	0.8406	55.1284	1.3603	6.9747	6.5775	249.8068	4.3283	60.2926	77.4996
	(0.0906)	(0.0163)	(0.4438)	(0.4552)	(2.4107)	(0.1229)	(2.5239)	(1.4679)	(9.8989)	(0.5846)	(1.8420)	(2.0432)
5	7.1838	0.7842	0.6925	0.6864	60.4738	2.6492	6.9871	5.5144	276.0558	9.5129	63.7258	79.3698
	(0.1030)	(0.0406)	(0.3678)	(0.3962)	(2.1240)	(0.3192)	(1.7397)	(1.4583)	(12.5507)	(1.4442)	(1.9378)	(1.4647)
6	7.2941	1.0282	0.6749	0.5540	65.4299	4.5061	6.9266	7.1030	301.8258	19.5153	63.4614	76.7321
	(0.0940)	(0.0650)	(0.4246)	(0.2860)	(1.3532)	(0.5389)	(1.9822)	(2.2783)	(19.3400)	(3.6895)	(9.4703)	(10.0277)

Table 4.4: Average CPU time in seconds of KDE, OPT, DET(L_2), and DET(MLE). The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a product of (independent) piecewise constant densities with 10 *discontinuities along each dimension.*

the essential histogram in the sense of capturing the important features (e.g. modes). Compared to DET(MLE), DET(L_2) tends to give an overly smoothed estimate of the density and is unable to pick out some the of lower density modes. FDE and OPT give a much tighter fit to the true density than the others by producing many more discontinuities in their estimates. However,when the underlying density is not smooth as in Figure (4.3), OPT gives multiple spurious modes, while both DET estimators and the essential histogram can recover the behavior of the true density most of the time. In addition, we also notice that both DET estimators occasionally produce density ”spikes”, most likely due to the greedy nature of the method.

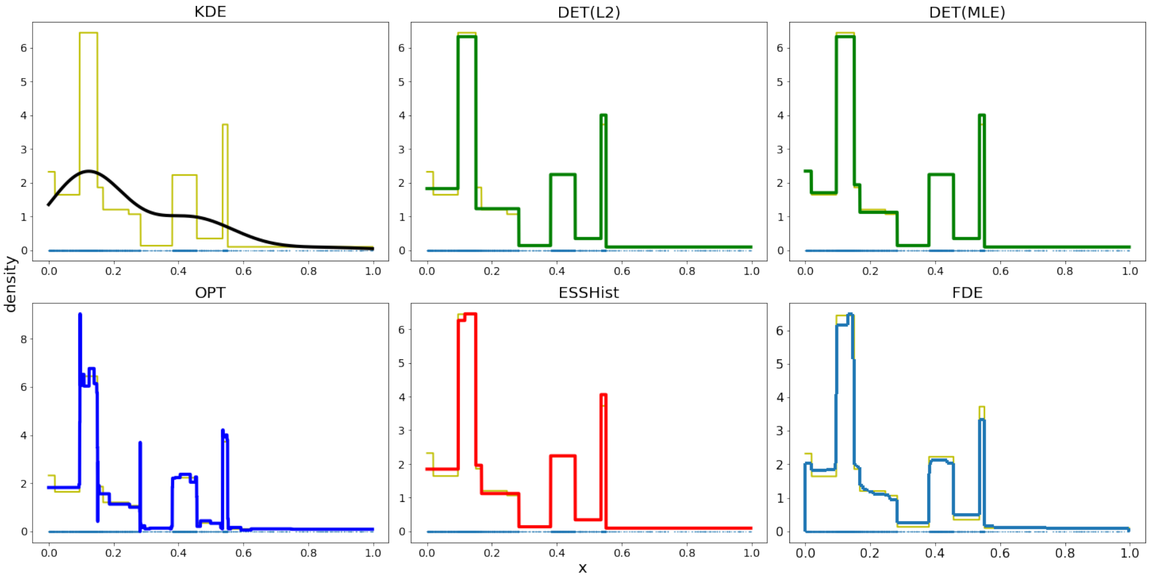
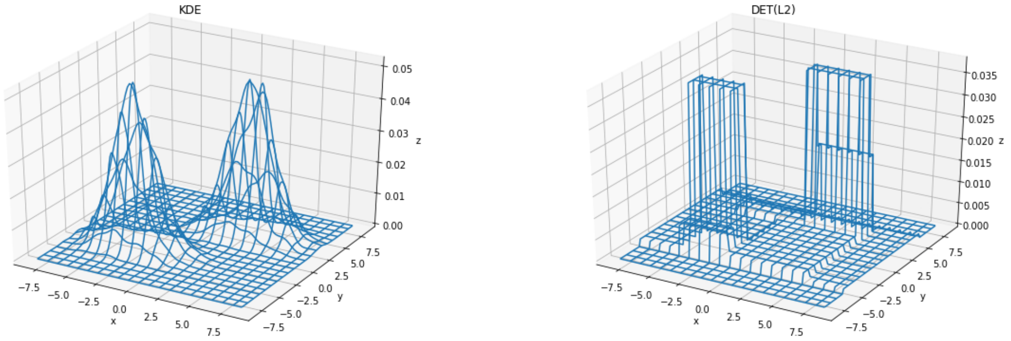


Figure 4.3: Density Estimates for KDE, DET(L_2),DET(MLE), OPT,Essential Histogram, and FDE when the underlying distribution is a univariate piecewise constant distribution with 10 *discontinuities. Sample size is 5000.*

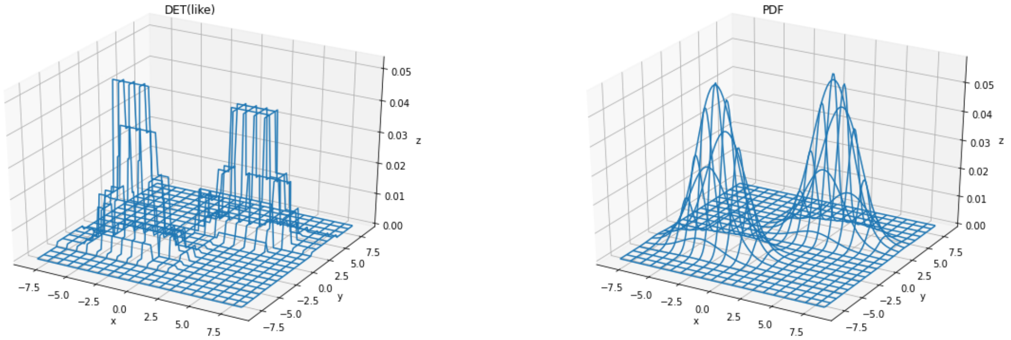
2D examples. Next, we compare the performance of KDE, DET(L_2) and DET(MLE) in two dimensions. Figure 4.4 gives the case when the underlying density is mixture of two Gaussian with the same covariance matrix, and 4.5 gives the case when the covariance matrices are different. The results are consistent with those of the multi-dimensional experiments. We have that KDE gives almost perfect recovery of the true density, while both DET estimators struggle. Still, DET(MLE) does a much better job then DET(L_2) at capturing the general shape of the density.

Figure 4.6 gives an example of a two dimensional piecewise constant density, where the number of discontinuities in the first dimension is 2, and the number of discontinuities in the second dimension in 4. Figure 4.7 gives another example of piecewise constant density where the density has 3 discontinuities in the first dimension and is constant in the other. In both examples, we notice that DET(L_2)



(a) KDE

(b) DET(L_2)

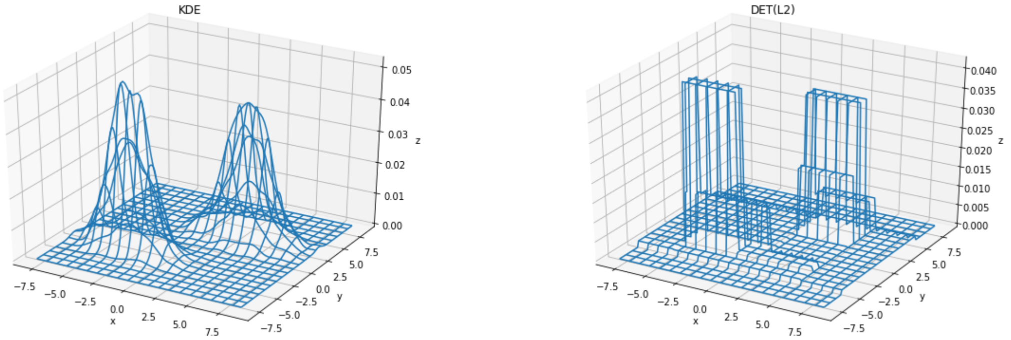


(c) DET(MLE)

(d) pdf

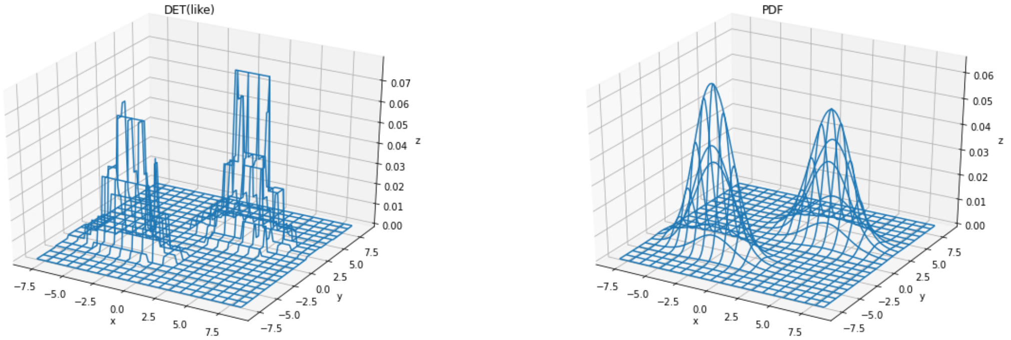
Figure 4.4: Density Estimates for KDE, DET(L_2), and DET(MLE) when the underlying distribution is a mixture of two gaussian distributions with the same covariance matrix. Sample size is 5000.

and DET(MLE) perform similarly to each other, both giving much more accurate estimates than KDE. Note that DET(L_2) is unable to pick out the low density region in Figure 4.6, which is, again, consistent with what we observe in the univariate examples.



(a) KDE

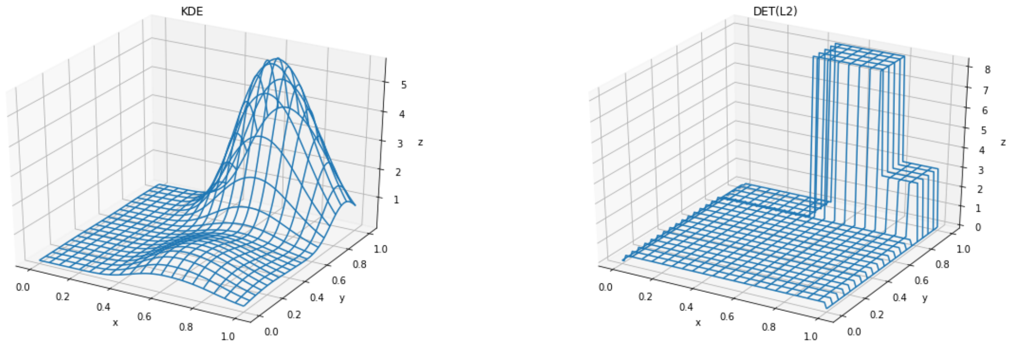
(b) DET(L_2)



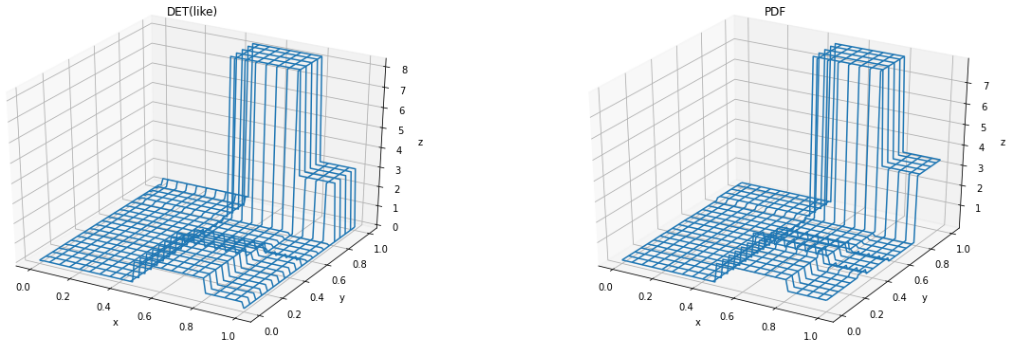
(c) DET(MLE)

(d) pdf

Figure 4.5: Density Estimates for KDE, DET(L_2), and DET(MLE) when the underlying distribution is a mixture of two gaussian distributions with different covariance matrices. Sample size is 5000.

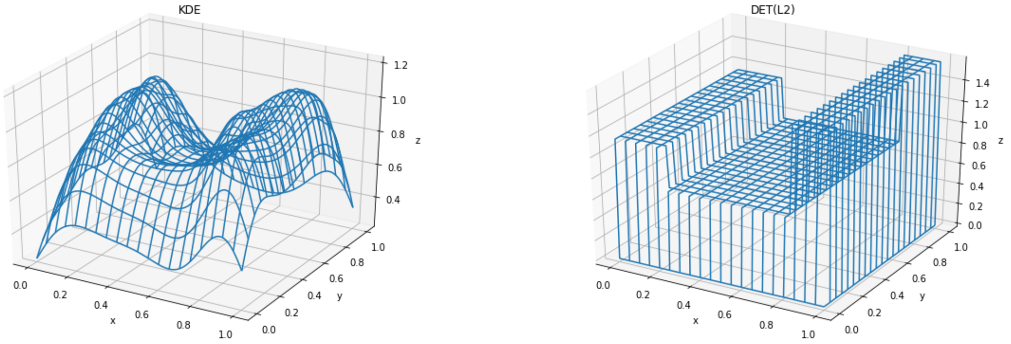


(a) KDE *(b) DET(L_2)*



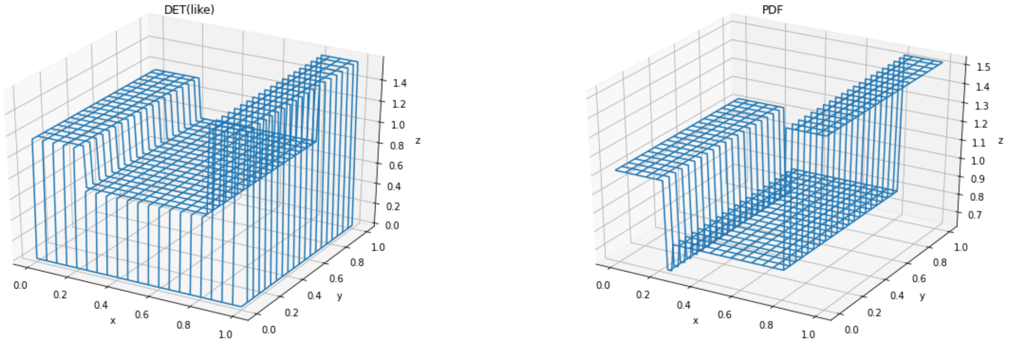
(c) DET(MLE) *(d) pdf*

Figure 4.6: Density Estimates for KDE, DET(L_2), and DET(MLE) when the underlying distribution is a piecewise constant distribution, with 2 splits on the x-axis and 4 splits on the y axis. Sample size is 5000.



(a) KDE

(b) DET(L_2)



(c) DET(MLE)

(d) pdf

Figure 4.7: Density Estimates for KDE, DET(L_2), and DET(MLE) when the underlying distribution is a piecewise constant distribution, with 3 splits on the x-axis and 0 splits on the y axis. Sample size is 5000.

Special cases. Finally, we give some more special examples where piecewise constant density estimates could be problematic. In Figure 4.8, we have a uniform density supported on two intertwined moon shaped domains. While KDE has little trouble with recovering the domain of the density and giving almost constant estimates, both DET methods fail in both ways. Since the moon shaped domain cannot be well approximated by axis-aligned rectangles, the domains given by the DET estimators are chunky and disconnected. The density estimates are also far from being constant. The bright spots on the heatmap indicate that the density estimates in those regions are significantly higher than the rest.

Figure 4.9 gives another similar example where the underlying density is supported on two concentric circles. The densities are constant on both rings, with the values on the outer ring being slightly lower than that of the inner ring. DET(L_2), in this case, completely misses the outer ring while DET(MLE) does a sightly better job at capturing the shape of the support.

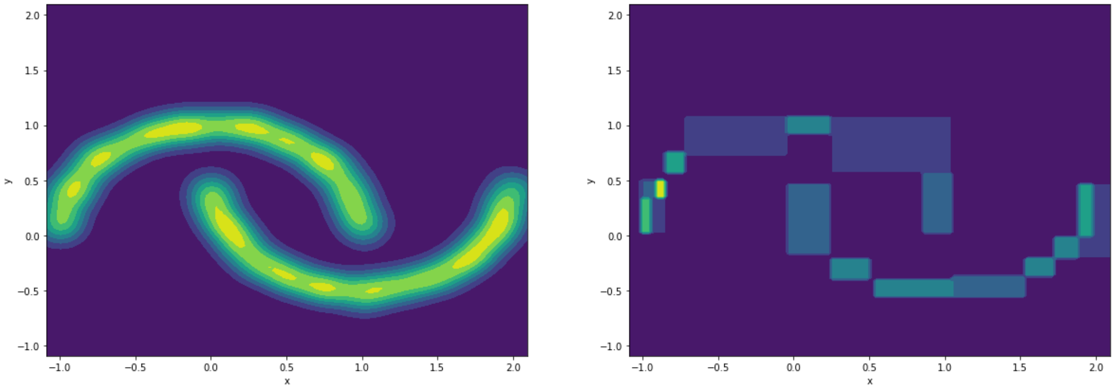
4.1.3 Interpretability

As mentioned in Ram and Gray (2011), DET(L_2) can be used to perform variable selection. The variable importance of a covariate a is defined as the total reduction in the surrogate loss over all nodes with a as the splitting,

$$\sum_{\Omega} (\widehat{R}(\Omega) - \widehat{R}(\Omega_L) - \widehat{R}(\Omega_R))\mathbb{I}(\Omega^{\text{split dim}} = a).$$

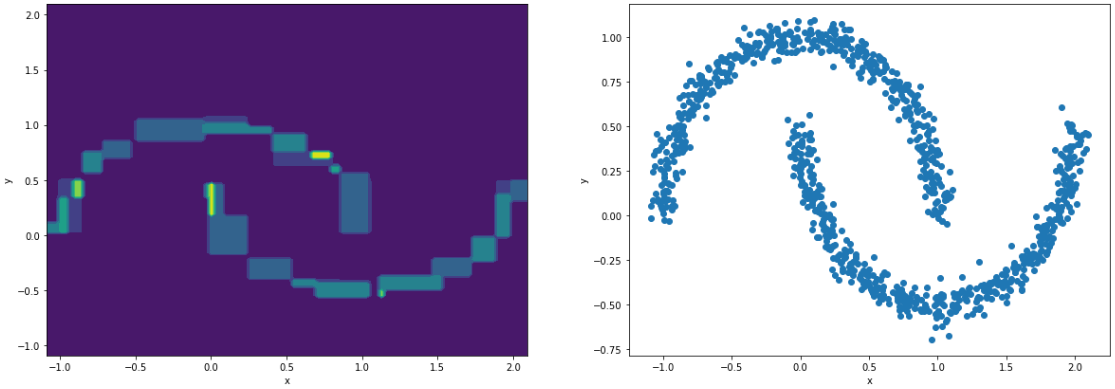
The authors demonstrate the application of variable importance on two real datasets and show that the irrelavant variables almost never get split on. The variable importance of a covariate a for DET(MLE) can be similarly defined.

We illustrate the performance of the two metrics with a simple toy example. Consider $X \in \mathbb{R}^{10}$, where $(X_1, X_2, X_3) \sim \mathcal{N}(\mu, \Sigma)$ for some μ and Σ, and $(X_4, \ldots, X_{10}) \sim$



(a) KDE

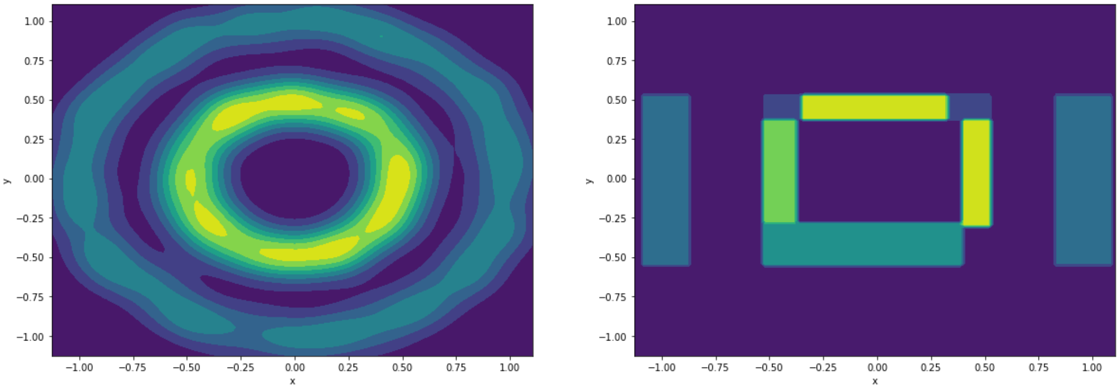
(b) DET(L_2)



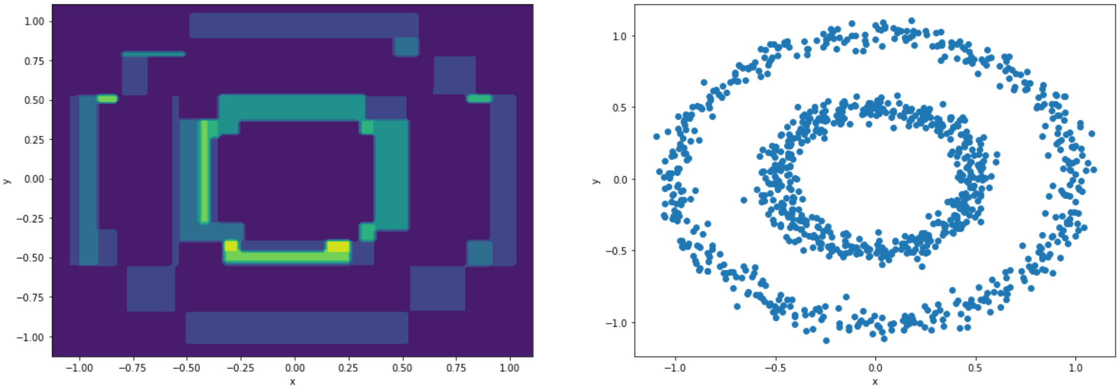
(c) DET(MLE)

(d) dataset

Figure 4.8: Contourplot of the density Estimates for KDE, DET(L_2), and DET(MLE) when the underlying distribution is a uniform density supported on two intertwined moon shaped domains.



(a) KDE *(b) DET(L_2)*



(c) DET(MLE) *(d) dataset*

Figure 4.9: Contourplot of the density Estimates for KDE, DET(L_2), and DET(MLE) when the underlying distribution is a uniform density supported on two concentric circles.

DET(L_2)	0.0025	0.0013	0.0023	0.0003	0	0	0	0.0002	0	0.0007
DET(MLE)	1121.37	843.86	929.47	0	0	0	0	0	0	0

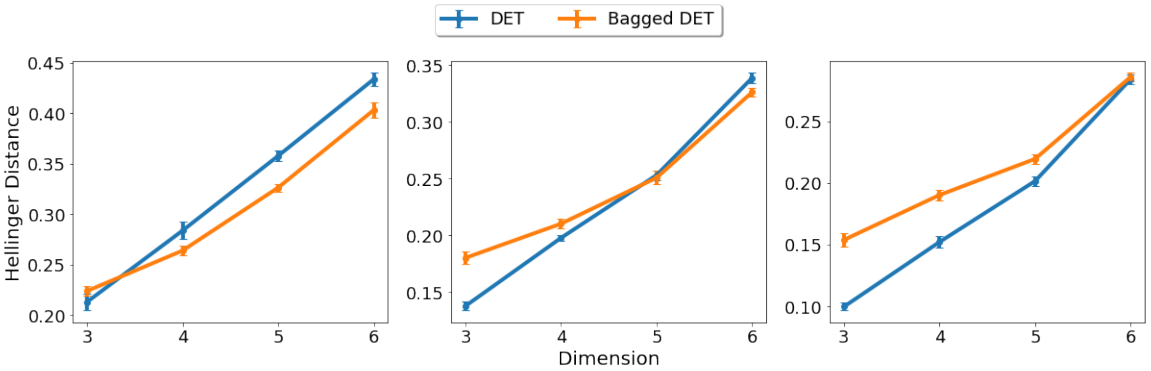
Table 4.5: Variable importance estimation of each covariate by DET(L_2) and DET(MLE) when the underlying distribution is $X \in \mathbb{R}^{10}$, where $(X_1, X_2, X_3) \sim \mathcal{N}(\mu, \Sigma)$ for some μ and Σ, and $(X_4, \ldots, X_{10}) \sim \mathrm{Unif}(0, 1)$

$\mathrm{Unif}(0, 1)$. We generate $N = 2000$ samples using this distribution and run DET(L_2) and DET(MLE) on the generated samples. The variable importance estimated by DET(L_2) for each covariate is given in Table 4.5.

Notice that both methods give their highest three variable importance values to the first three covariates. DET(MLE) does not split on the other seven covariates at all, while DET(L_2) would split on them very rarely.

4.2 BAGGING

Next, we demonstrate the effect of bagging on the DET(MLE) estimator. Table (4.6) and (4.7) records the error of the DET(MLE) and the bagged DET(MLE) in terms of hellinger distance for various dimensions and sample sizes when the underlying distribution is a mixture of four gaussian distributions and piecewise constant, respectively. The number of estimators for the bagged DET is taken to be 10 and the `MTRY` parameter is set to be $\sqrt{d}$ rounded down, a common choice when ensembling regression trees . Based on the results, it appears that, as least for the set of parameters we are using, bagging does not seem to provide significant improvement over the a single density tree. In fact, for large sample sizes, bagging actually worsen the performance of the estimators.



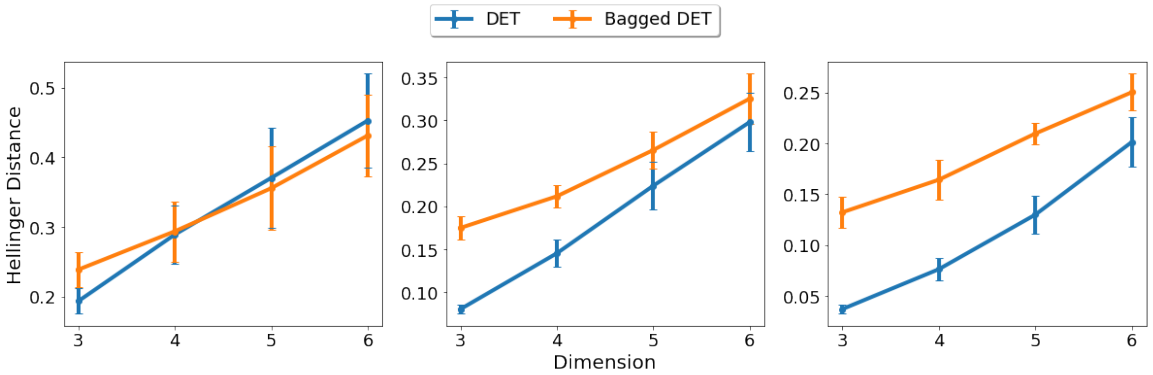
	N = 2000		N = 20000		N = 100000	
D	DET	Bagged DET	DET	Bagged DET	DET	Bagged DET
3	0.2132	0.2240	0.1377	0.1801	0.0996	0.1538
	(0.0085)	(0.0049)	(0.0040)	(0.0056)	(0.0031)	(0.0057)
4	0.2838	0.2640	0.1977	0.2102	0.1521	0.1902
	(0.0086)	(0.0049)	(0.0026)	(0.0045)	(0.0046)	(0.0044)
5	0.3577	0.3263	0.2530	0.2507	0.2015	0.2195
	(0.0051)	(0.0039)	(0.0043)	(0.0060)	(0.0037)	(0.0041)
6	0.4335	0.4030	0.3387	0.3261	0.2842	0.2860
	(0.0063)	(0.0078)	(0.0044)	(0.0035)	(0.0037)	(0.0033)

Table 4.6: Error in Hellinger Distance between the true density and DET and Bagged DET. The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a mixture of four Gaussian distributions.

4.3 DET with Confidence

In this section, we demonstrate the performance of all the DET-CF estimators. To show the efficiency and scalability of DET methods, we first compare them with the regular DET in terms of estimation error and running time. Next we provide some concrete lower dimensional illustrations of the DET-CF estimators.

All DET-CF methods are implemented in `C++` with bindings to Python and R. Please refer to Section 4.1 for more implementation details. Unless otherwise specified, for all our experiments, we are using the default model parameters, i.e. `MAX_LEAF_SIZE`



	$N = 2000$		$N = 20000$		$N = 100000$	
D	DET	Bagged DET	DET	Bagged DET	DET	Bagged DET
3	0.1934	0.2387	0.0804	0.1749	0.0372	0.1323
	(0.0181)	(0.0250)	(0.0054)	(0.0137)	(0.0047)	(0.0152)
4	0.2890	0.2937	0.1454	0.2117	0.0764	0.1643
	(0.0420)	(0.0432)	(0.0162)	(0.0127)	(0.0114)	(0.0196)
5	0.3704	0.3556	0.2240	0.2655	0.1300	0.2097
	(0.0714)	(0.0602)	(0.0281)	(0.0218)	(0.0183)	(0.0106)
6	0.4527	0.4312	0.2981	0.3253	0.2014	0.2503
	(0.0673)	(0.0585)	(0.0343)	(0.0290)	(0.0240)	(0.0182)

Table 4.7: Error in Hellinger Distance between the true density and DET, Bagged DET. The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a multivariate piecewise constant distribution.

$= 10$, `MIN_LEAF_SIZE` $= 5$, `MTRY` $= D$, the number of folds $K = 10$, confidence level `delta` $= 0.01$ and the list of `alpha` values to be the set of all distinct values from the training DET. In addition, we only consider the case of `criterion` being the KL divergence in this experiment, because it is the best performing estimator based on previous experiments. We'll denote DET(MLE) as DET in this section for simplicity.

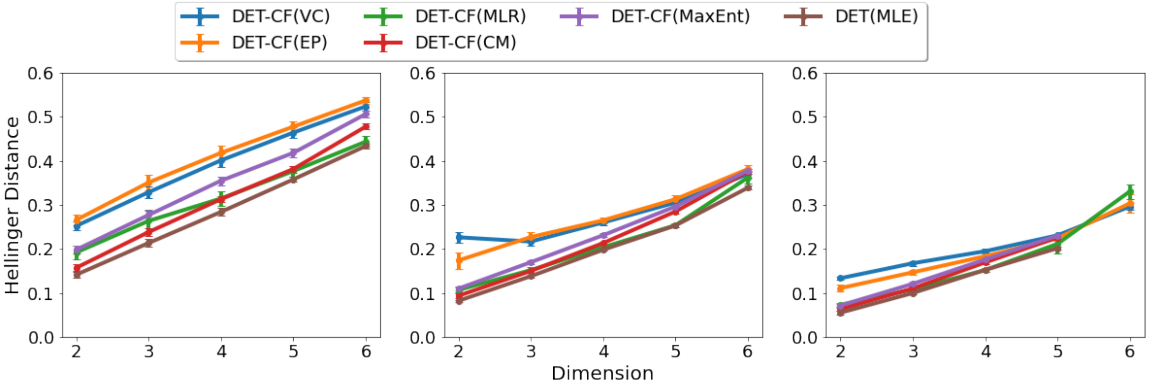
4.3.1 Accuracy and Speed

First, we compare the performance of all the DET-CF estimators with that of a regular DET in terms of Hellinger distance and running time when the underlying distribution is a mixture of four gaussians and piecewise constant, respectively. Please refer to Section 4.1 for more details regarding how the underlying distributions are generated.

Mixture of Gaussians

Table 4.8 records the average hellinger distance between the true density and the estimation from each DET-CF method over 20 replications. Table 4.2 records the average CPU time of each DET-CF method in seconds over the 20 replications. The numbers in the parenthesis are the corresponding standard deviation. The underlying distributions are identical to the ones generated in Section 4.1 and one can directly compare the numbers here with Table 4.1 and Table 4.2. In general, all DET-CF methods give very similar error rate as DET(MLE), with DET(MLE) giving the lowest error rate by a small margin in almost all cases. This is expected because the DET-CF methods are not designed to optimize any error metric, but rather be a simple estimator that belongs to a certain confidence set. Among the DET-CF methods, DET-CF(CM) and DET-CF(MLR) give the best estimation error overall, followed by DET-CF(MaxEnt) and the other two greedy methods. The confidence set constructed using the multiscale likelihood ratio test (3.1.8) appears to be a better choice for density estimation.

In terms of computational efficiency, the three greedy confidence methods DET-CF(VC), DET-CF(EP) and DET-CF(MLR) give faster computation than DET(MLE), especially for large sample sizes. DET-CF(CM) and DET-CF(MaxEnt) are computed using interior point solvers and require significantly more computational time than the greedy methods.



	$N = 2000$					$N = 20000$					$N = 100000$				
D	VC	EP	MLR	CM	MaxEnt	VC	EP	MLR	CM	MaxEnt	VC	EP	MLR	CM	MaxEnt
2	0.2519	0.2655	0.1918	0.1569	0.1979	0.2262	0.1736	0.1065	0.0933	0.1095	0.1332	0.1107	0.0727	0.0629	0.0713
	(0.0100)	(0.0118)	(0.0160)	(0.0077)	(0.0090)	(0.0115)	(0.0185)	(0.0064)	(0.0048)	(0.0048)	(0.0043)	(0.0074)	(0.0033)	(0.0034)	(0.0044)
3	0.3286	0.3507	0.2634	0.2380	0.2768	0.2165	0.2266	0.1515	0.1496	0.1695	0.1672	0.1466	0.1078	0.1099	0.1205
	(0.0129)	(0.0171)	(0.0227)	(0.0083)	(0.0123)	(0.0101)	(0.0114)	(0.0067)	(0.0039)	(0.0043)	(0.0054)	(0.0057)	(0.0057)	(0.0036)	(0.0038)
4	0.4010	0.4179	0.3143	0.3128	0.3547	0.2598	0.2647	0.2036	0.2130	0.2308	0.1946	0.1832	0.1517	0.1692	0.1758
	(0.0157)	(0.0173)	(0.0162)	(0.0069)	(0.0101)	(0.0064)	(0.0066)	(0.0027)	(0.0026)	(0.0039)	(0.0049)	(0.0068)	(0.0048)	(0.0033)	(0.0040)
5	0.4640	0.4775	0.3766	0.3813	0.4180	0.3050	0.3126	0.2534	0.2847	0.2961	0.2306	0.2242	0.2112	0.2256	0.2299
	(0.0130)	(0.0125)	(0.0118)	(0.0077)	(0.0090)	(0.0082)	(0.0085)	(0.0045)	(0.0056)	(0.0049)	(0.0038)	(0.0064)	(0.0210)	(0.0036)	(0.0026)
6	0.5235	0.5372	0.4434	0.4776	0.5062	0.3697	0.3811	0.3618	0.3757	0.3778	0.2963	0.3034	0.3306	*	*
	(0.0106)	(0.0075)	(0.0122)	(0.0067)	(0.0069)	(0.0079)	(0.0098)	(0.0139)	(0.0062)	(0.0037)	(0.0068)	(0.0206)	(0.0166)	*	*

Table 4.8: Error in Hellinger Distance between the true density and DET-CF(VC), DET-CF(EP), DET-CF(MLR), DET-CF(CM), and DET-CF(MaxEnt). The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a mixture of four Gaussian distributions.

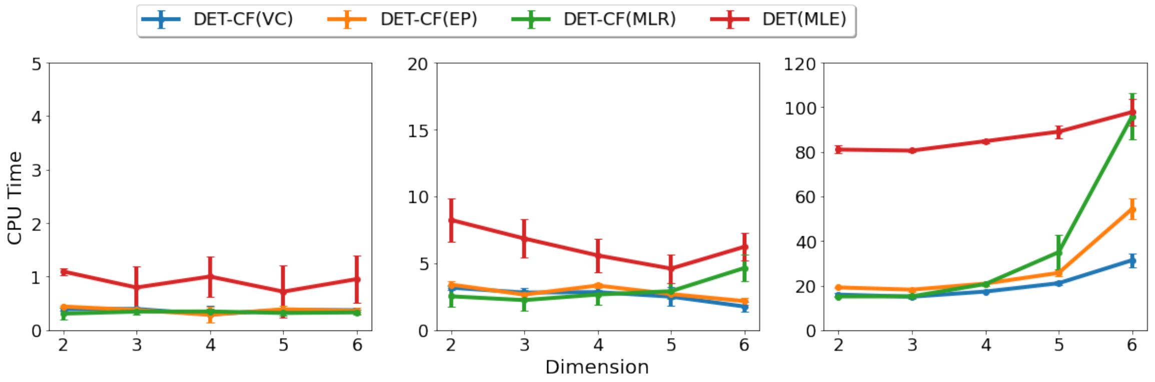
Piecewise Constant Densities

Similar to the Gaussian scenario, Table 4.10 and Table 4.11 give the estimation error and running time when the data are generated from piecewise constant densities. Again, the results recorded are directly comparable with those in Table 4.3 and Table 4.4. We observe similar behaviors between the DET-CF estimators and DET(MLE) as in the Gaussian case.

4.3.2 Adaptability

Next, we show some concrete examples in lower dimensions.

1D Examples. In this example, we compare the performance of DET(MLE) with the five DET-CF estimators. Figure 4.10, 4.11, and 4.12 give the case when the underlying



			$N = 2000$					$N = 20000$					$N = 100000$		
D	VC	EP	MLR	CM	MaxEnt	VC	EP	MLR	CM	MaxEnt	VC	EP	MLR	CM	MaxEnt
2	0.3913	0.4413	0.3082	3.2874	4.1148	3.1747	3.4063	2.5281	21.8085	43.2803	15.9269	19.1556	15.1119	158.5226	412.8647
	(0.0309)	(0.0218)	(0.1162)	(1.3856)	(0.6786)	(0.1728)	(0.2625)	(0.7416)	(6.4673)	(7.8530)	(0.8553)	(0.3190)	(0.3801)	(10.2557)	(44.0395)
3	0.3962	0.3760	0.3467	2.5563	2.7266	2.8082	2.6542	2.2438	12.8603	61.9080	14.9962	18.1550	15.1476	147.6845	826.1208
	(0.0213)	(0.0498)	(0.0557)	(0.8921)	(0.4967)	(0.3693)	(0.2726)	(0.7630)	(5.0141)	(16.6778)	(0.7091)	(0.8634)	(0.9140)	(6.2034)	(160.3396)
4	0.2922	0.2871	0.3491	3.4075	4.7722	2.8407	3.3347	2.6602	23.2326	87.3465	17.2904	20.8743	20.6650	166.3502	1274.8135
	(0.1577)	(0.1520)	(0.0553)	(1.6381)	(0.5003)	(0.1976)	(0.2001)	(0.7734)	(8.8571)	(10.6239)	(0.3121)	(0.3822)	(0.8092)	(8.2125)	(138.6695)
5	0.3784	0.3899	0.3206	1.4910	3.3637	2.5040	2.6932	2.9124	18.9044	133.2233	21.0483	25.6754	34.9552	179.1269	2065.9864
	(0.0262)	(0.0699)	(0.0420)	(0.4892)	(0.6706)	(0.7120)	(0.2620)	(0.5698)	(6.5240)	(33.4005)	(0.9136)	(1.3374)	(7.8037)	(5.6049)	(355.6428)
6	0.3725	0.3592	0.3317	3.3571	5.5570	1.7656	2.1700	4.6472	18.4917	178.9235	31.3642	54.3841	95.8679	*	*
	(0.0190)	(0.0551)	(0.0472)	(1.7107)	(1.5081)	(0.3824)	(0.2520)	(1.0050)	(5.0183)	(19.3438)	(3.0306)	(4.6970)	(10.2990)	*	*

Table 4.9: Average CPU time in seconds of DET-CF(VC), DET-CF(EP), DET-CF(MLR), DET-CF(CM), and DET-CF(MaxEnt). The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a mixture of four Gaussian distributions.

densities are generated from the claw distribution (mixture of 5 Gaussian distributions with unequal mean and same variance), harp distribution (mixture of 5 Gaussian distributions with unequal mean and variance), and a piecewise constant distribution with 10 discontinuities respectively. The number of samples is 5000 in each case.

In general, all five DET-CF estimators are doing decent work, with DET-CF(MLR) and DET-CF(CM) giving the best fit overall since they are able to capture the modes of the underlying distribution in almost all cases. In comparison, DET-CF(MaxEnt) also does a good job at picking out the modes, but due to the nature of using the entropy as the objective function, it tends to smoothen out the relative magnitude of the density estimates for the modes. DET-CF(VC) and DET-CF(EP), on the other hand, are constructed using more conservative confidence sets thatn DET-CF(MLR), and

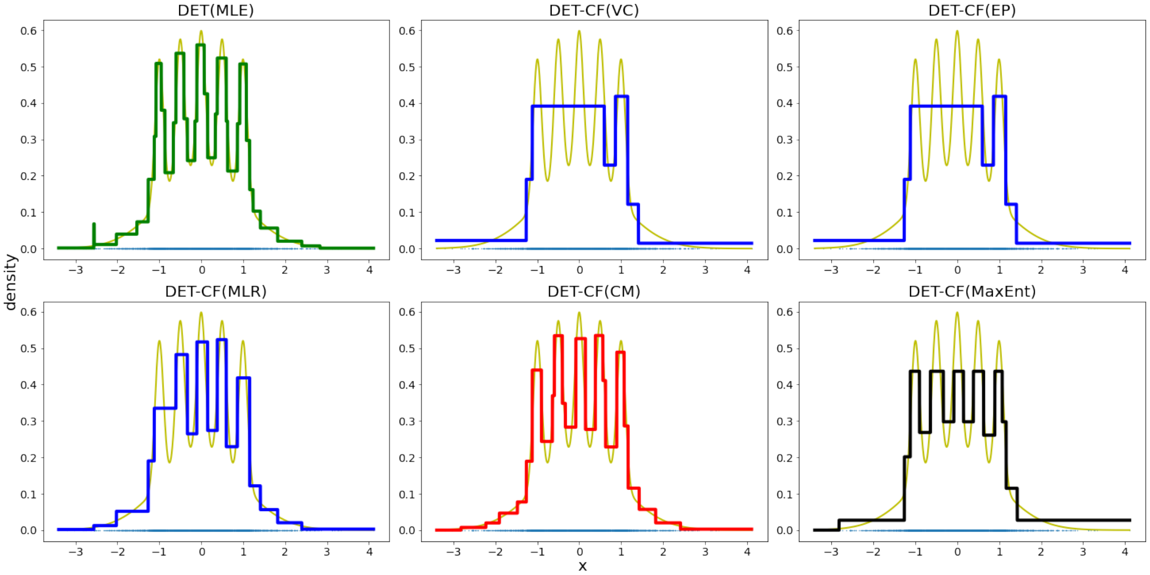


Figure 4.10: Density Estimates for DET(MLE), DET-CF(VC),DET-CF(EP), DET-CF(MLR), DET-CF(CM), and DET-CF(MaxEnt) when the underlying distribution is a univariate claw distribution. Sample size is 5000.

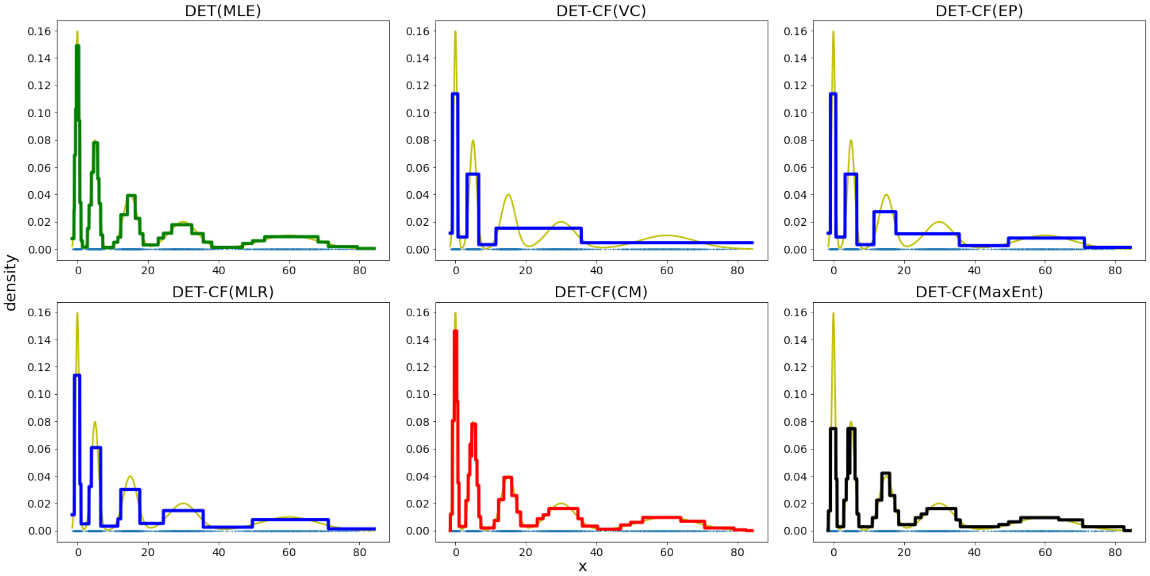
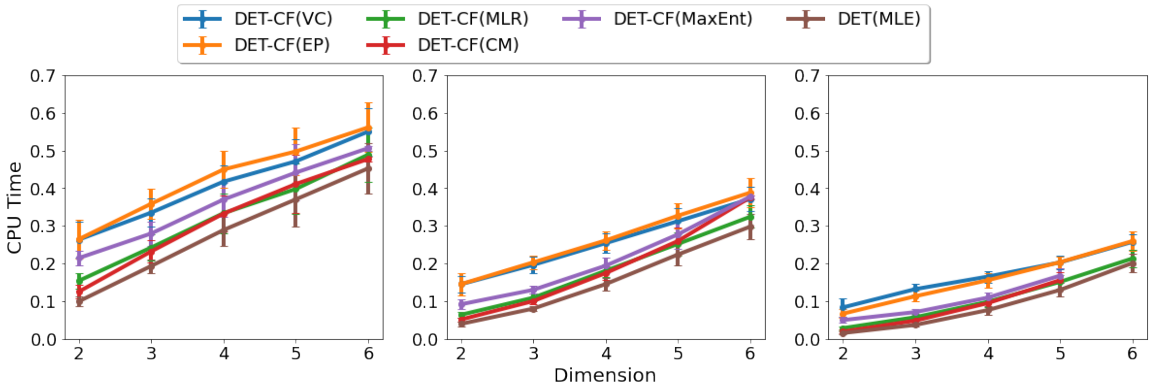


Figure 4.11: Density Estimates for DET(MLE), DET-CF(VC),DET-CF(EP), DET-CF(MLR), DET-CF(CM), and DET-CF(MaxEnt) when the underlying distribution is a univariate harp distribution. Sample size is 5000.



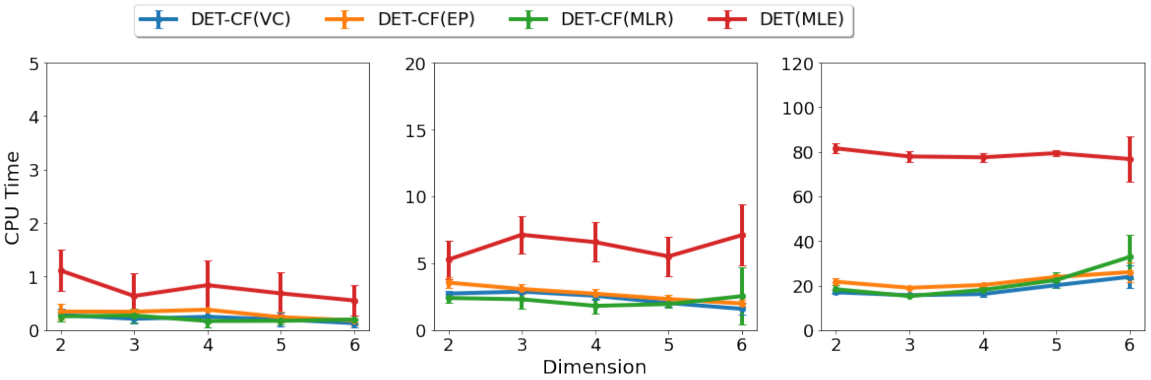
	$N = 2000$					$N = 20000$					$N = 100000$				
D	VC	EP	MLR	CM	MaxEnt	VC	EP	MLR	CM	MaxEnt	VC	EP	MLR	CM	MaxEnt
2	0.2634	0.2645	0.1542	0.1252	0.2146	0.1450	0.1453	0.0640	0.0514	0.0919	0.0836	0.0670	0.0287	0.0195	0.0500
	(0.0487)	(0.0518)	(0.0206)	(0.0179)	(0.0199)	(0.0212)	(0.0292)	(0.0087)	(0.0051)	(0.0136)	(0.0242)	(0.0121)	(0.0054)	(0.0055)	(0.0079)
3	0.3351	0.3585	0.2424	0.2321	0.2797	0.1965	0.2032	0.1094	0.1004	0.1302	0.1322	0.1133	0.0576	0.0487	0.0712
	(0.0370)	(0.0390)	(0.0349)	(0.0289)	(0.0324)	(0.0222)	(0.0180)	(0.0099)	(0.0081)	(0.0105)	(0.0138)	(0.0138)	(0.0087)	(0.0061)	(0.0071)
4	0.4174	0.4497	0.3336	0.3325	0.3699	0.2541	0.2613	0.1807	0.1746	0.1942	0.1654	0.1557	0.0984	0.0950	0.1096
	(0.0442)	(0.0495)	(0.0532)	(0.0481)	(0.0454)	(0.0254)	(0.0238)	(0.0170)	(0.0187)	(0.0207)	(0.0156)	(0.0192)	(0.0133)	(0.0151)	(0.0130)
5	0.4714	0.4975	0.3979	0.4111	0.4414	0.3125	0.3270	0.2522	0.2599	0.2774	0.2031	0.2037	0.1514	0.1558	0.1679
	(0.0585)	(0.0626)	(0.0665)	(0.0770)	(0.0748)	(0.0355)	(0.0340)	(0.0301)	(0.0369)	(0.0372)	(0.0172)	(0.0158)	(0.0171)	(0.0212)	(0.0194)
6	0.5503	0.5615	0.4891	0.4776	0.5062	0.3715	0.3883	0.3243	0.3757	0.3778	0.2575	0.2597	0.2135	*	*
	(0.0630)	(0.0652)	(0.0723)	(0.0067)	(0.0069)	(0.0330)	(0.0386)	(0.0297)	(0.0062)	(0.0037)	(0.0211)	(0.0256)	(0.0226)	*	*

Table 4.10: Error in Hellinger Distance between the true density and DET-CF(VC), DET-CF(EP), DET-CF(MLR), DET-CF(CM), and DET-CF(MaxEnt). The numbers in parentheses are standard errors from 20 replicas.The underlying distribution is a product of (independent) piecewise constant densities with 10 *discontinuities along each dimension.*

therefore are unable to detect the modes well enough for this sample size. Increasing the sample size will, however, eventually allow both methods to detect all modes in the underlying distributions. Note that compared to the original DET methods, the DET-CF methods no longer produce any ”spikes” in the density estimates.

4.4 SUMMARY

In summary, in this chapter, we provided extensive numerical simulations of the methods considered in Chpater 2 and Chapter 3. We started off with a thorough comparison of KDE, OPT, DET(L_2), and DET(MLE) in terms of estimation error in hellinger distance and CPU time over a variety of high dimensional scenarios. Our



	$N = 2000$					$N = 20000$					$N = 100000$				
D	VC	EP	MLR	CM	MaxEnt	VC	EP	MLR	CM	MaxEnt	VC	EP	MLR	CM	MaxEnt
2	0.2844	0.3480	0.2575	3.1801	2.5931	2.7328	3.5543	2.4030	23.9594	29.7355	16.9600	21.7133	18.3225	161.2779	179.6800
	(0.1216)	(0.1492)	(0.0947)	(1.5589)	(0.9430)	(0.2192)	(0.3906)	(0.3231)	(6.2444)	(4.6218)	(0.8481)	(1.6986)	(1.7127)	(5.0472)	(6.5356)
3	0.2122	0.3450	0.2740	2.4087	3.0155	2.8780	3.0719	2.3041	16.5161	32.8860	15.6460	18.9827	15.2751	151.3886	329.2281
	(0.0955)	(0.0308)	(0.1283)	(1.1814)	(1.3716)	(0.1530)	(0.3400)	(0.6946)	(9.8668)	(9.4952)	(0.5179)	(0.4174)	(0.2650)	(4.7503)	(66.4500)
4	0.2494	0.3818	0.1706	0.9347	3.0250	2.5694	2.7218	1.8154	9.1920	76.3758	16.1812	20.2446	18.0513	144.4974	815.6579
	(0.1149)	(0.0207)	(0.1209)	(0.2499)	(1.0981)	(0.3037)	(0.3655)	(0.5458)	(1.1914)	(33.0447)	(1.0610)	(1.0484)	(1.0783)	(13.3569)	(220.0140)
5	0.1997	0.2430	0.1744	1.3924	3.3234	2.0261	2.3263	1.9489	19.4516	152.8674	20.1490	23.8465	22.4505	158.4581	1667.0901
	(0.1245)	(0.1070)	(0.0719)	(0.6007)	(1.7818)	(0.3007)	(0.2746)	(0.2638)	(7.6965)	(47.6800)	(1.1857)	(1.2385)	(3.6469)	(11.3878)	(433.7138)
6	0.1310	0.1819	0.1989	3.3571	5.5570	1.5912	1.9942	2.5400	18.4917	178.9235	23.9152	26.0299	32.8056	*	*
	(0.0733)	(0.0855)	(0.0680)	(1.7107)	(1.5081)	(0.4361)	(0.2640)	(2.1239)	(5.0183)	(19.3438)	(5.0058)	(4.4507)	(9.8502)	*	*

Table 4.11: Average CPU time in seconds of DET-CF(VC), DET-CF(EP), DET-CF(MLR), DET-CF(CM), and DET-CF(MaxEnt). The numbers in parentheses are standard errors from 20 replicas. The underlying distribution is a product of (independent) piecewise constant densities with 10 *discontinuities along each dimension.*

results show that DET(MLE) gives a consistent improvement over DET(L_2) in terms of estimation error, while only being marginally slower in computational time. Both DET methods show a big advantage over KDE in terms of computational efficiency, and also in terms of estimation error when the underlying distribution is not smooth. Compared to LL-OPT (the fast implementation of OPT), DET(MLE) is a bit lacking in computational efficiency, especially for large sample sizes, due to the intricacy of the cross-validation step. However, DET(MLE) enjoys slightly better estimation error than OPT. Our findings are further backed by various low dimensional illustrations. Notably, a potential drawback of the DET methods is that they would occasionally overfit by producing a spike in the density estimates.

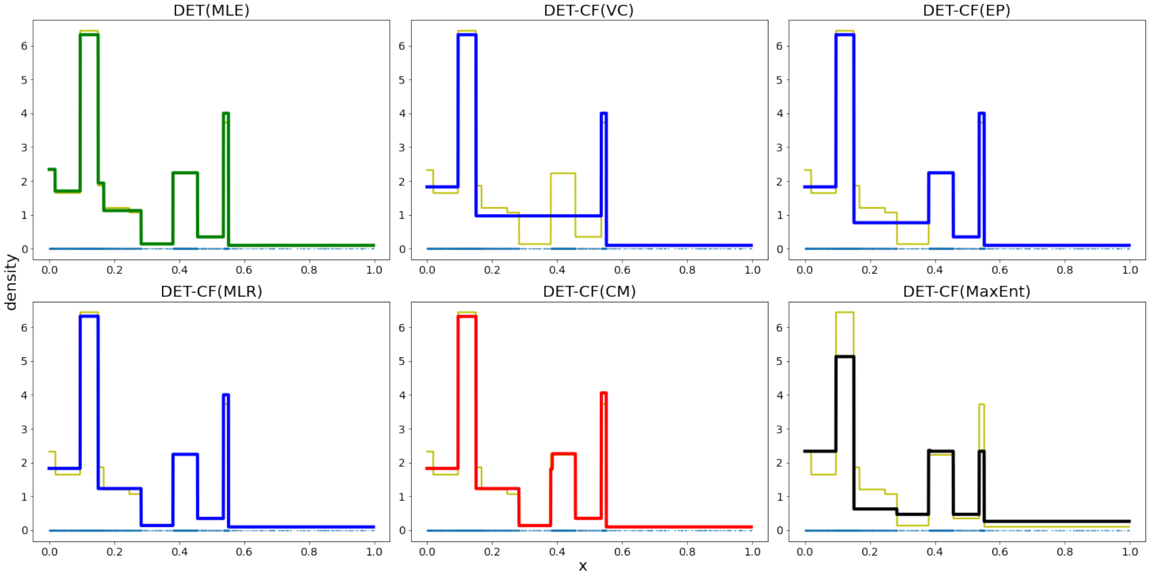


Figure 4.12: Density Estimates for DET(MLE), DET-CF(VC),DET-CF(EP), DET-CF(MLR), DET-CF(CM), and DET-CF(MaxEnt) when the underlying distribution is a univariate piecewise constant distribution with 10 *discontinuities. Sample size is 5000.*

Next, we investigate the effectiveness of bagging with DET estimators. For the set of model parameters we considered in our experiments, we did not observe an improvement of bagged DET over DET. In fact, the estimation error is generally higher with bagging.

Finally, we compare the numerical performance of the five DET with confidence methods introduced in Chapter 3 with that of the DET(MLE). All method give very similar estimation performances, with DET(MLE) giving the lowest error, followed closely by DET-CF(MLR) and DET-CF(CM). The greedy DET-CF methods also proves to be notably more computationally efficient than DET(MLE) whereas the exact DET-CF methods solved using constrained optimization are significantly slower. Our low dimensional illustrations show that the confidence set constructed using the multiscale likelihood ratio test gives the sharpest confidence set out of all three methods. Notably, DET-CF(MLR) and DET-CF(CM) are both able to capture the descriptive features well and, unlike the DET(MLE), they do not produce any spikes in the density estimates.

Five

Applications

Density estimation is a powerful tool in numerous applications. In this section, we demonstrate some of them.

5.1 Level Set Trees

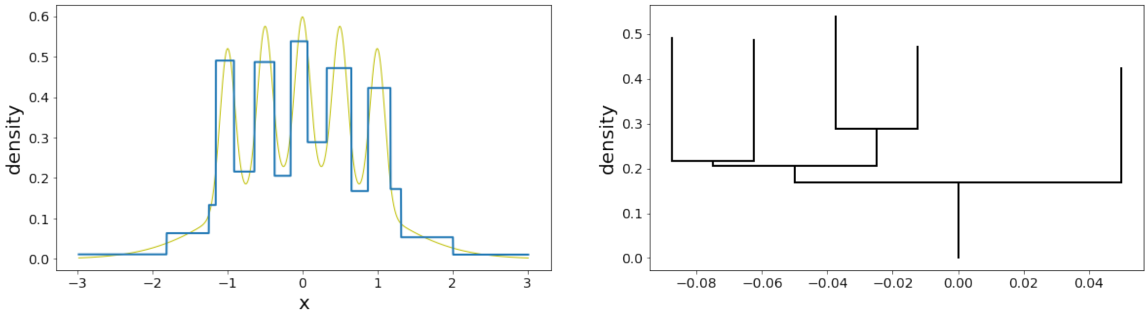
The level set tree of a probability density function is a useful tool for visualizing and presenting the hierarchy of the modes of the function, and sees applications in cluster analysis, statistical inference of the density estimates, function optimization, etc. In particular, the level set tree, when applied as a cluster analysis tool, can provide a more informative and holistic visualization of the data topography compared to many traditional clustering algorithms. For example, classical clustering methods such as the K-means Macqueen (1967), Lloyd (1982) and spectral clustering Shi and Malik (2000) rely on knowing the number of clusters K a priori. While effective in certain cases, such an assumption can be problematic when the samples are very noisy or corrupted, or exhibit complex multimodal behavior and spatial heterogeneity, or simply when the true number of clusters is unknown. As a result, hierarchical clustering becomes a preferable method in those scenarios because its construction does not require prior knowledge on the number of clusters K. In addition, the dendrogram provides a more informative depiction of the clustering structure for various cluster levels. However, the result of the dendrogram can be highly susceptible to the choice of the linkage criteria Hartigan (1975), Hastie et al. (2001), and computing the dendrogram involves storing the $n \times n$ pairwise distance matrix for the samples, which can be computationally expensive for large scale problems.

The level set tree provides a good alternative way of visualizing the cluster structure of the data through density estimation. More formally, let $L_t = \{x : p(x) > t\}$ denote an upper level set of p. For a given level t, L_t may be decomposed into finitely many disjoint sets: $L_t = \cup_{i=1}^{m} C_i$ for some m. The sets $C_1, \ldots, C_m$ are the denoted as the level set clusters at level t. The tree structure of the level sets comes from the fact that for levels $t_1 \neq t_2$, if C_1 is a level set cluster at level t_1 and C_2 a level set cluster at level t_2, then we have either (i) $C_1 \subset C_2$ or (ii) $C_2 \subset C_1$, or (iii) $C_1 \cap C_2 = \emptyset$.

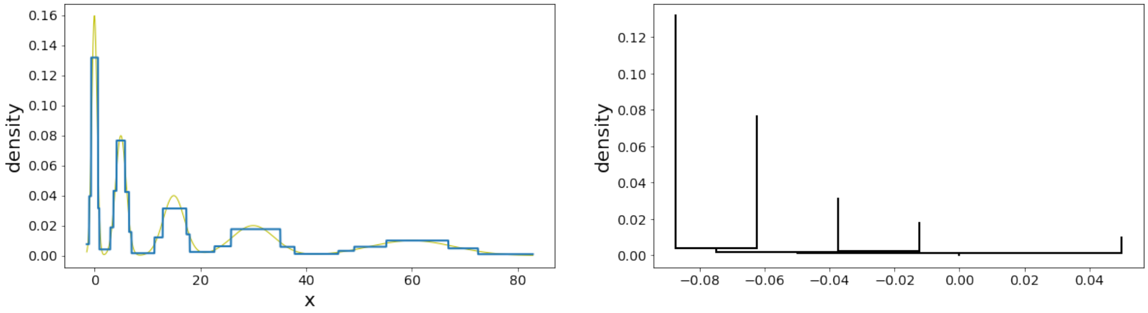
In general, constructing the level set tree for a smooth density function can be challenging because it is almost impossible to recover the level set exactly for each level t. Instead, the level set clusters are generally estimated with data using graph-based algorithms Bobrowski and Kahle (2018). For example, `DeBaCl` Kent et al. (2013) is a `Python` package for constructing level set trees which utilizes k-nearest neighbor graphs for estimating the clusters.The level set tree estimation for smooth density estimates also relies on a high resolution of cluster levels t for a representative tree.

However, on the other hand, constructing the level set tree for a piecewise constant density is almost immediate. Because there are only finitely many density values for each estimated density, we only need to look at finitely many cluster levels t. In addition, the support of the piecewise constant densities being axis-aligned rectangles means that we can provide exact recovery of the level set clusters based on the estimated densities. Finally, on top of all else, for tree-based density estimation methods such as DET, the tree structure allows extremely fast querying of the density tree, which in turn allows a extremely efficient level set tree construction. This makes tree based density estimation methods especially desirable in high dimensions, since they provide a simple tool for visualizing the cluster structure in high dimensions.

Finally, we demonstrate the effectiveness of the level set tree for DET(MLE) on two simple univariate examples: the claw distribution and the harp distribution. The results are given in Figure 5.1. We see that in both examples, the level set tree does a good job at detecting the modes of the underlying density.



(a) Claw



(b) Harp

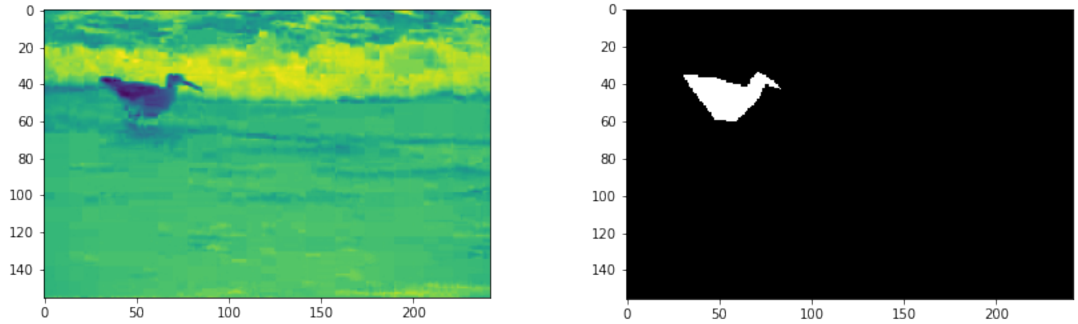
Figure 5.1: Demonstration of the Level Set Tree for DET(MLE) on (a) univariate claw distribution (b) univariate harp distribution. Figure on the left show s the underlying density function and the DET(MLE) estimates. Figure on the right plots the level set tree.

5.2 Background Subtraction

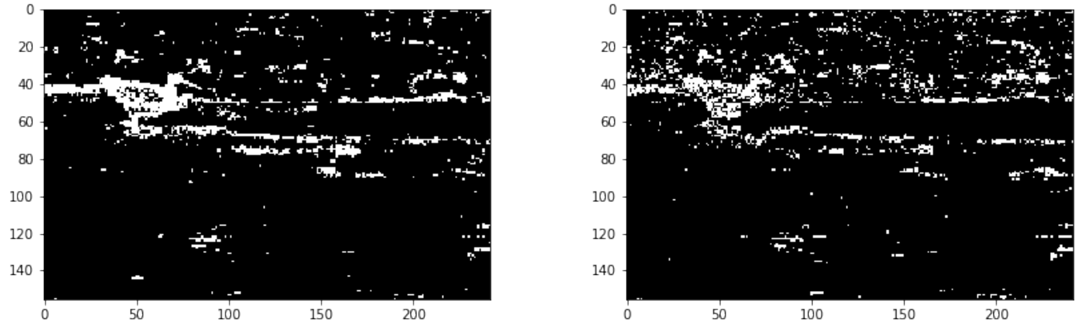
Background subtraction is one of the major tasks in the area of computer vision and image processing whose aim is to detect changes in a video data sequence. In many applications, one does not need to have all the information about the evolution of movement in the video sequence, but rather only requires the information of the changes in the scene, because an image's regions of interest are objects (humans, animals, vehicles, etc) in the foreground.

Density estimation can be applied in background subtraction through the estimation of the probability of observing pixel intensity values based on a sample of intensity values for each pixel. Effective density estimators can adapt quickly to changes in the images, which enables very sensitive detection of moving targets. For example, Elgammal et al. (2000) illustrates how KDE can be applied an a background subtraction method; the authors discuss how to estimate the bandwidth matrix of KDE, and introduce a false detection suppression technique as a post-processing step. Jang et al. (2008) explores using the oriental histogram Jang et al. (2007) as the density estimator. The method is shown to be relatively robust to changes in illumination and small movements. The authors show that the method can be further sped up by incorporating an integral histogram Porikli (2005) technique.

In both examples we mentioned above, the bandwidth matrix for KDE and the number of bins for the histogram are fixed. Next, as a simple illustration, we consider the performance of the DET(MLE) and KDE applied in background subtraction and demonstrate how adaptive density estimators can potentially provide better results. Figure 5.2 shows an example of a video data sequence with a total of 76 image frames. We take one of them as the test frame (shown in Figure 5.2a), and use the remaining



(a) Data *(b) Label*



(c) DET(MLE) *(d) KDE*

Figure 5.2: Application in background Subtraction: comparison of DET(MLE) and KDE.

as the training data for the density estimators. The label for the foreground (a duck) is shown in Figure 5.2b. Figure 5.2c and 5.2d show, respectively, the detected foreground of the DET(MLE) and kDE estimators. The threshold density values for classifying the foreground are chosen as the ones that yield the maximum average precision scores. As we can see, in this example, DET(MLE) gives a much clearer foreground recovery than KDE.

5.3 Anomaly Detection

Anomaly detection is the process of detecting instances that deviate significantly from the other sample members. The problem of detecting anomalies can arise in many different applications, such as fraud detection in financial transactions, intrusion detection for security systems, and various medical examinations. In Gu et al. (2019) we study various anomaly detection methods under the unsupervised setup, where we do not assume any prior knowledge on the label of the normal and anomalous instances.

Many empirical methods have been developed in the unsupervised setup, and can be roughly classified into four categories: density based methods such as the Robust KDE (RKDE) Kim and Scott (2012), Local Outlier Factor (LOF) Breunig et al. (2000), and mixture models (EGMM); distance based methods such as kNN Angiulli and Pizzuti (2002) and Angle-based Outlier Detection (ABOD) Li et al. (2015); model based methods such as the one-class SVM (OCSVM) Schölkopf et al. (2001), SVDD Tax and Duin (2004), and autoencoders Chen et al. (2017); ensemble methods such as Isolation Forest (IForest) Liu et al. (2008), LODA Pevný (2016), and PIDForest Gopalan et al. (2019).

Through our investigation of various anomaly detection methods, we noticed that essentially all methods boils down to constructing a ”score” for measuring the degree of anomaly of the instances. A good scoring function would give more extreme values to anomalous instances than normal ones, so that one can easily tell them apart based on the score values. For example, kNN Angiulli and Pizzuti (2002) uses the average k nearest neighbor distance as the score for measuring anomalies. Anomalous instances are expected to be further away than normal ones, and thus have higher kNN distance values. IForest Liu et al. (2008) constructs a set of trees by sequentially randomly partitioning the space. Each node in the tree corresponds to a sector of the partition. The score of a instance is set to be the average path lengths from the root to the leave nodes that the instance belongs to. As anomalies tend to be outlying and scarce, they are more likely to be isolated faster by the random partitioning than normal instances, therefore leading to shorter average path lengths.

It is not hard to notice that the scores in anomaly detection have a density flavor to them, as anomalies are expected to be rare and outlying, they usually lie in low density regions as opposed to normal instances. Therefore, a good density estimator can potentially be a good anomaly detector. In addition, many benchmark and real datasets in anomaly detection are rather large and high dimensional, which makes efficient density based anomaly detection methods that scales well with sample size and dimension more appealing.

In this section, we apply DET(MLE) and KDE to a variety of anomaly detection real datasets from the ODDS library . For simple illustration purposes, we are taking the estimated density values from the density estimators as the anomaly scores. We understand that such a choice is not necessarily ideal because density values does not take into consideration some important factors, such as the location of the sample points. In practice, more complicated anomaly scores are generally used.

In the next experiment, we compare the performance of DET(MLE) and KDE with that of IForest and LOF, two state-of-the-art anomaly detection methods, in terms of AUC and average precision (AP) scores. The results are given in Table 5.1 and 5.2. Our results show that, even though both DET(MLE) and KDE do not perform as well as the other two anomaly detectors, they are actually giving very

Data	n	d	IForest	LOF	DET(MLE)	KDE
arrhythmia	452	274	0.7914	0.7614	0.6851	0.5539
cardio	1831	21	0.9205	0.7013	0.5879	0.5884
ionosphere	351	33	0.8496	0.9023	0.5649	0.9199
lympho	148	18	1.0000	0.9824	0.8087	0.9683
musk	3062	166	0.9999	0.2860	0.5000	0.0997
pima	768	8	0.6697	0.5528	0.6208	0.5511
satellite	6435	36	0.7058	0.5787	0.6433	0.5660
satimage-2	5803	36	0.9923	0.9915	0.9889	0.6456
thyroid	3772	6	0.9762	0.9630	0.9574	0.9330
vowels	1456	12	0.7499	0.9373	0.8546	0.8647
WBC	278	30	0.9474	0.9025	0.9188	0.8616
avg.rank			0.4545	1.2727	1.9091	2.3636

Table 5.1: AUC scores for the IForest, LOF, DET(MLE) and KDE on various anomaly detection real datasets from the ODDS library.

comparable performance in many examples. In particular, DET(MLE) outperform KDE in most cases, because being data adaptive makes it better at picking out modes and irregularities of the density. In addition, the decent computational efficiency allows DET(MLE) to potentially take on more computationally heavy tasks, which would be a challenge for distance-based methods such as LOF and KDE.

5.4 CLASSIFICATION

Finally, we illustrate the performance of DET estimators applied in classification problems. The procedure is standard. Given labeled data $(X_1, Y_1), \ldots, (X_n, Y_n)$ where Y_i's are the corresponding labels belonging to k classes $\{C_1, \ldots, C_k\}$. For a new sample X^*, we'd like to predict the label Y^* associated with X^*. The Bayes classifier selects

Data	n	d	IForest	LOF	DET(MLE)	KDE
arrhythmia	452	274	0.4445	0.3641	0.3608	0.1611
cardio	1831	21	0.5816	0.2002	0.1459	0.1708
ionosphere	351	33	0.8030	0.8698	0.5180	0.9117
lympho	148	18	1.0000	0.7695	0.4665	0.5900
musk	3062	166	0.9958	0.0217	0.0317	0.0169
pima	768	8	0.4969	0.3852	0.4526	0.3788
satellite	6435	36	0.6556	0.4068	0.4778	0.3517
satimage-2	5803	36	0.9383	0.5104	0.4596	0.0182
thyroid	3772	6	0.5084	0.3975	0.2560	0.2232
vowels	1456	12	0.1572	0.3989	0.2075	0.6729
WBC	278	30	0.6137	0.2654	0.2958	0.4194
avg.rank			0.4545	1.4545	2.0000	2.0909

Table 5.2: Average Precision scores for the IForest, LOF, DET(MLE) and KDE on various anomaly detection real datasets from the ODDS library.

the label that gives the maximal posterior probability

$$\begin{aligned} & \underset{C_i}{\operatorname{argmax}} \mathbb{P}(C_i|X_1, \ldots, X_n) \\ = & \underset{C_i}{\operatorname{argmax}} \mathbb{P}(X_1, \ldots, X_n|C_i)\mathbb{P}(C_i) \end{aligned}$$

where the equality follows from the Bayes rule. Replacing $\mathbb{P}(C_i)$ with its the plug-in estimator $\frac{1}{n}\sum_{j=1}^{n} \mathbb{I}(X_j \in C_i)$ and $\mathbb{P}(X_1, \ldots, X_n|C_i)$ with density estimates on those samples with a class label C_i yields an estimate of the class label.

Table 5.3 records the classification accuracy rates of random forest, kernel support vector machine (SVM), logistics regression, KDE, and DET(MLE) applied on three real datasets: the Higgs boson machine learning challenge dataset ATLAS collaboration (2014), the Cherenkov imaging gamma-ray telescope MAGIC dataset Heck et al. (1998), Frank and Asuncion (2010), and a letter recognition dataset Frank and Asuncion

(2010). In all examples, we randomly sample $\sim 80\%$ of the data as training data for constructing the classifiers. The remaining data are taken as the testing data for obtaining classification accuracy rates.

The Higgs ATLAS collaboration (2014) dataset has $818,238$ observations and 35 features, where each observation is a simulated proton-proton collision event in the official ATLAS full detector simulator. A detailed description of the features can be found in ATLAS collaboration (2014). We follow the same preprocessing procedure as in Chakravarti et al. (2021) (see their Section 5.1 for more details) and arrive at a total of $165,027$ observations with 2 classes ($80,806$ background observations and $84,221$ signal observations) and 15 features. Table 5.3 shows that random forest and kernel SVM give the highest classification rates. The performance of DET(MLE) is close to logistic regression and KDE, but not as good as random forest and kernel SVM.

MAGIC Heck et al. (1998), Frank and Asuncion (2010) is a set of simulated data from a physics-based model for the gamma-ray Cherenkov telescope. The dataset consists of a total of $19,020$ observations with 2 classes (6688 background observations and $12,332$ signal observations) and 10 features. The signal and background observations are, respectively, images of hadronic showers caused by gamma rays vs other cosmic rays in the upper atmosphere. Table 5.3 shows that random forest, kernel SVM, KDE and DET(MLE) all demonstrate similarly compelling performances.

The letter Frank and Asuncion (2010) dataset is a discrete 16-dimensional dataset with $20,000$ observations. The original raw dataset records a large number of black-and-white rectangular pixel displays coming from 26 capital letters in the English alphabet. Each display is converted into 16 primitive numerical attributes that are

Table 5.3: Classification rates for the Higgs, Letter, and MAGIC data: Random Forest, Kernel SVM, Logistic Regression, KDE and DET(MLE)

Dataset	Random Forest	Kernel SVM	Logistic Regression	KDE	DET(MLE)
Higgs	0.7452	0.7346	0.6285	0.6495	0.6334
MAGIC	0.8276	0.8815	0.7759	0.8329	0.8446
Letter	0.8288	0.9412	0.6930	*	0.9179

then scaled to fit into a range of integer values from 0 through 15. Table 5.3 shows that both kernel SVM and DET(MLE) dominate the other methods by giving over 90% classification accuracy rates. Note the KDE is omitted in this example because the dataset is discrete.

5.5 SUMMARY

In summary, in this chapter, we study several applications of density estimation. We argue that the computational efficiency of querying a DET allows for fast construction of its level set tree, which can be a valuable asset in downstream analyses such as cluster analysis. We demonstrate the effectiveness of level set tree construction with DET using two simple examples. We explain how density estimation can be applied in background subtraction and illustrate the performance of DET(MLE) and KDE over video stream data. We consider the application of DET(MLE) and KDE in anomaly detection (using the density values as the anomaly scores) and provide comparison studies with two other state-of-the-art anomaly detectors over a selection of real datasets. Finally, we compare the performance of applying DET(MLE) as a bayes classifier with other popular classification methods over three large scale high dimensional datasets.

Six

Software

The original DET(L_2) method developed by Ram and Grey is implemented in `C++` as part of the `mlpack` library under namespace `mlpack::det` . The `mlpack` library is an open source machine learning library in `C++` with `CLI` support and bindings with `Python`, `R`, `Julia`, etc. DET(MLE) is developed as a complement to the DET(L_2) method by adding in `criterion` as an extra argument in the `det` function in `mlpack`. DET-CF is developed as an additional function to the `mlpack` library. The complete documentation of the two functions is given as follows.

6.1 Installation

Since the DET (with `criterion` as an option) and DET-CF functions are not developed as part of the released functions in the `mlpack` library, they need to be added to the library and built manually.

- Download the latest version of `mlpack` from `https://github.com/mlpack/mlpack`.
- Include the source code for DET and DET-CF `https://github.com/guxiaoyi/DET-var` under the unpacked directory `'src/mlpack/methods'`.

- Build the `mlpack` library from source by following the instructions provided in `https://www.mlpack.org/doc/stable/doxygen/build.html`. Note that it is not necessary to install the `mlpack` library to the system. You may run the bindings in the build directory by specifying the correct path for the bindings.

6.2 FUNCTIONS

The `det` function implements DET(L_2) and DET(MLE), and the `det_cf` function implements DET-CF(VC), DET-CF(EP), and DET-CF(MLR). The arguments of `det` is given as follows.

6.2.1 `det`

Arguments

`train` (-t)	The data set on which to build a density estimation tree.
`test` (-T)	A set of test points to estimate the density of.
`folds` (-f)	The number of folds of cross-validation to perform for the estimation (0 is LOOCV). Default value 10.
`criterion` (-r)	The loss function used for growing the tree. Possible choices: "L2", "NLL". Default value "NLL".
`max_leaf_size` (-L)	The maximum size of a leaf in the unpruned, fully grown DET. Default value 10.
`min_leaf_size` (-l)	The minimum size of a leaf in the unpruned, fully grown DET. Default value 5.
`mtry` (-R)	The number of features considered for splitting the tree node (0 is equivalent to all features). Default value 0.
`input_model` (-m)	A trained density estimation tree.
`verbose` (-v)	Display informational messages and the full list of parameters and timers at the end of execution.

Values

`test_set_estimates` (`-E`)	The density estimates on the test set from the final optimally pruned tree.
`training_set_estimates` (`-e`)	The ensity estimates on the training set from the final optimally pruned tree.
`output_model` (`-M`)	Trained density estimation tree to.
`train_time` (`-Q`)	Training time.
`query_time` (`-q`)	Querying time.
`vi` (`-i`)	The variable importance values for each feature.
`upper` (`-P`)	The upper bounding box of the hyper-rectangles in the support of the trained DET.
`lower` (`-W`)	The lower bounding box of the hyper-rectangles in the support of the trained DET.
`dens` (`-D`)	The density values associated with each hyper-rectangle in the support of the trained DET.

Examples

The following gives a command-line example of fitting a DET(MLE) estimator with 10-fold cross-validation, where the training data is supplied in the file `train.csv` and the density estimates on the test data is stored in `test.csv`.

```
mlpack_det -t train.csv -T test.csv -f 10 -r NLL
```

The following gives a `Python` example of fitting a DET(L_2) estimator with `mtry` = 3. `dtree` is a dictionary containing all the output values.

```
import mlpack

from mlpack import det

dtree = det(train = trainData, test = testData, criterion = ’L2’, mtry = 3)

testVals = dtree[’test_set_estimates’]
```

6.2.2 `det_cf`

Arguments

`train` (`-t`)	The data set on which to build a density estimation tree as an inital estimator.
`val` (`-e`)	The data set on which to generate the confidence statements.
`test` (`-T`)	A set of test points to estimate the density of.
`folds` (`-f`)	The number of folds of cross-validation to perform for the estimation (0 is LOOCV). Default value 10.
`criterion` (`-r`)	The loss function used for growing the tree. Possible choices: "L2", "NLL". Default value "NLL".
`method` (`-a`)	The type of confidence statements used. Possible choices: "vc", "ep", "mlr". Default value "mlr".
`alpha_list` (`-p`)	The list of alpha values (upper level set probability content) for generating the level set based confidence statements. Default is using all distinct alpha values from the training DET.
`delta` (`-d`)	Confidence level associated with the confidence statements. Default value 0.01.
`max_leaf_size` (`-L`)	The maximum size of a leaf in the unpruned, fully grown DET. Default value 10.
`min_leaf_size` (`-l`)	The minimum size of a leaf in the unpruned, fully grown DET. Default value 5.
`mtry` (`-R`)	The number of features considered for splitting the tree node (0 is equivalent to all features). Default value 0.
`input_model` (`-m`)	A trained density estimation tree.
`verbose` (`-v`)	Display informational messages and the full list of parameters and timers at the end of execution.

Values

test_set_estimates (-E)	The density estimates on the test set from the final optimally pruned tree.
training_set_estimates (-e)	The ensity estimates on the training set from the final optimally pruned tree.
output_model (-M)	Trained density estimation tree to.
train_time (-Q)	Training time.
query_time (-q)	Querying time.
vi (-i)	The variable importance values for each feature.
upper (-P)	The upper bounding box of the hyper-rectangles in the support of the trained DET.
lower (-W)	The lower bounding box of the hyper-rectangles in the support of the trained DET.
dens (-D)	The density values associated with each hyper-rectangle in the support of the trained DET.

Examples

The following gives a command-line example of fitting a DET-CF(MLR) estimator with confidence level 0.95, where the training data is supplied in the file `train.csv`, validation data is supplied in the file `val.csv`, and the density estimates on the test data is stored in `test.csv`.

```
mlpack_det_cf -t train.csv -T -e val.csv test.csv -a mlr -d 0.05
```

The following gives a `Python` example of fitting a DET-CF(VC) estimator with `delta` being a grid of values between 0 and 1 (exclusive). `dtree` is a dictionary containing all the output values.

```
import mlpack
from mlpack import det_cf
```

```
deltas = np.linspace(0.1,0.9,9)

dtree = det_cf(train = trainData, test = testData, method = ’vc’, delta = deltas)

testVals = dtree[’test_set_estimates’]
```

Bibliography

Angiulli, F. and Pizzuti, C. (2002). Fast outlier detection in high dimensional spaces. In Principles of Data Mining and Knowledge Discovery, pages 15–27, Berlin, Heidelberg. Springer Berlin Heidelberg.

ATLAS collaboration (2014). Dataset from the atlas higgs boson machine learning challenge 2014. CERN Open Data Portal.

Bassett, R. and Sharpnack, J. (2019). Fused density estimation: theory and methods. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 81(5):839–860.

Berger, A. L., Pietra, V. J. D., and Pietra, S. A. D. (1996). A maximum entropy approach to natural language processing. Comput. Linguist., 22(1):39–71.

Beygelzimer, A., Kakade, S., and Langford, J. (2006). Cover trees for nearest neighbor. In Proceedings of the 23rd International Conference on Machine Learning, ICML '06, page 97–104, New York, NY, USA. Association for Computing Machinery.

Bickel, P. J. and Rosenblatt, M. (1973). On Some Global Measures of the Deviations of Density Function Estimates. The Annals of Statistics, 1(6):1071 – 1095.

Bobrowski, O. and Kahle, M. (2018). Topology of random geometric complexes: a survey. Journal of Applied and Computational Topology, 1.

Bousquet, O., Boucheron, S., and Lugosi, G. (2004). Introduction to Statistical Learning Theory, pages 169–207. Springer Berlin Heidelberg, Berlin, Heidelberg.

Breiman, L., Friedman, J., Stone, C., and Olshen, R. (1984). Classification and Regression Trees. The Wadsworth and Brooks-Cole statistics-probability series.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000). Lof: Identifying density-based local outliers. SIGMOD Rec., 29(2):93–104.

Chakravarti, P., Kuusela, M., Lei, J., and Wasserman, L. (2021). Model-independent detection of new physics signals using interpretable semi-supervised classifier tests.

Chaudhuri, K., Dasgupta, S., Kpotufe, S., and von Luxburg, U. (2014). Consistent procedures for cluster tree estimation and pruning. IEEE Transactions on Information Theory, 60:7900–7912.

Chen, J., Sathe, S., Aggarwal, C. C., and Turaga, D. S. (2017). Outlier detection with autoencoder ensembles. In SDM.

Chernozhukov, V., Chetverikov, D., and Kato, K. (2013). Gaussian approximations and multiplier bootstrap for maxima of sums of high-dimensional random vectors. The Annals of Statistics, 41(6):2786 – 2819.

Chernozhukov, V., Chetverikov, D., and Kato, K. (2016). Comparison and anti-concentration bounds for maxima of Gaussian random vectors. CeMMAP working papers CWP40/16, Centre for Microdata Methods and Practice, Institute for Fiscal Studies.

Della Pietra, S., Della Pietra, V., and Lafferty, J. (1997). Inducing features of random fields. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(4):380–393.

Dudík, M., Phillips, S. J., and Schapire, R. E. (2004). Performance guarantees for regularized maximum entropy density estimation. In Shawe-Taylor, J. and Singer, Y., editors, Learning Theory, pages 472–486, Berlin, Heidelberg. Springer Berlin Heidelberg.

Dudík, M. (2011). Maximum entropy density estimation and modeling geographic distribution of species.

Einmahl, U. and Mason, D. M. (2005). Uniform in bandwidth consistency of kernel-type function estimators. The Annals of Statistics, 33(3):1380 – 1403.

Elgammal, A., Harwood, D., and Davis, L. (2000). Non-parametric model for background subtraction. In Vernon, D., editor, Computer Vision — ECCV 2000, pages 751–767, Berlin, Heidelberg. Springer Berlin Heidelberg.

Ferguson, T. S. (1973). A Bayesian Analysis of Some Nonparametric Problems. The Annals of Statistics, 1(2):209 – 230.

Frank, A. and Asuncion, A. (2010). Uci machine learning repository. `http://archive.ics.uci.edu/ml`.

Giné, E. and Guillou, A. (2002). Rates of strong uniform consistency for multivariate kernel density estimators. Annales de l'Institut Henri Poincare (B) Probability and Statistics, 38(6):907–921.

Gopalan, P., Sharan, V., and Wieder, U. (2019). Pidforest: Anomaly detection via partial identification. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, Advances in Neural Information Processing Systems 32, pages 15809–15819. Curran Associates, Inc.

Gramacki, A. and Gramacki, J. (2017). Fft-based fast computation of multivariate kernel density estimators with unconstrained bandwidth matrices. Journal of Computational and Graphical Statistics, 26(2):459–462.

Gray, A. G. and Moore, A. W. (2003). Nonparametric density estimation: Toward computational tractability.

Gu, X., Akoglu, L., and Rinaldo, A. (2019). Statistical analysis of nearest neighbor methods for anomaly detection. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, Advances in Neural Information Processing Systems 32, pages 10923–10933. Curran Associates, Inc.

Hartigan, J. A. (1975). Clustering Algorithms. John Wiley & Sons, Inc., USA, 99th edition.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). The Elements of Statistical Learning. Springer New York Inc., New York, NY, USA.

Heck, D., Knapp, J., Capdevielle, J. N., Schatz, G., and Thouw, T. (1998). CORSIKA: a Monte Carlo code to simulate extensive air showers.

Jang, D., Chai, Y., Jin, X., and Kim, T. (2007). Realtime coarse pose recognition using a multi-scaled local integral histograms. In 2007 International Conference on Convergence Information Technology (ICCIT 2007), pages 1982–1987.

Jang, D., Jin, X., Choi, Y., and Kim, T. (2008). Background subtraction based on local orientation histogram. In Lee, S., Choo, H., Ha, S., and Shin, I. C., editors, Computer-Human Interaction, pages 222–231, Berlin, Heidelberg. Springer Berlin Heidelberg.

Jaynes, E. T. (1957). Information theory and statistical mechanics. Phys. Rev., 106:620–630.

Jiang H, Mu JC, Y. K. D. C. L. L. W. W. (2016). Computational aspects of optional pólya tree. Journal of computational and graphical statistics : a joint publication of American Statistical Association, Institute of Mathematical Statistics, Interface Foundation of North America, 25(1):301–320.

Kapur, J. N. and Kesavan, H. K. (1992). Entropy Optimization Principles and Their Applications, pages 3–20. Springer Netherlands, Dordrecht.

Karl, P. (1895). Contributions to the mathematical theory of evolution.—ii. skew variation in homogeneous material. 186:343–414.

Kent, B., Rinaldo, A., and Verstynen, T. (2013). Debacl: A python package for interactive density-based clustering.

Kim, J. and Scott, C. D. (2012). Robust kernel density estimation. J. Mach. Learn. Res., 13(1):2529–2565.

Lee, D. and Gray, A. G. (2008). Fast high-dimensional kernel summations using the monte carlo multipole method.

Lei, J., G'Sell, M., Rinaldo, A., Tibshirani, R., and Wasserman, L. (2016). Distribution-free predictive inference for regression. Journal of the American Statistical Association, 113.

Li, H., Munk, A., Sieling, H., and Walther, G. (2020). The essential histogram. Biometrika, 107(2):347–364.

Li, X., Lv, J. C., and Cheng, D. (2015). Angle-based outlier detection algorithm with more stable relationships. In Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Volume 1, pages 433–446, Cham. Springer International Publishing.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, pages 413–422, Washington, DC, USA. ICDM '08.

Liu, H., Lafferty, J., and Wasserman, L. (2007). Sparse nonparametric density estimation in high dimensions using the rodeo. volume 2, pages 283–290, San Juan, Puerto Rico. Proceedings of Machine Learning Research.

Lloyd, S. (1982). Least squares quantization in pcm. IEEE Transactions on Information Theory, 28(2):129–137.

Lu, L., Jiang, H., and Wong, W. H. (2013). Multivariate density estimation by bayesian sequential partitioning. Journal of the American Statistical Association, 108(504):1402–1410.

Mack, Y. and Rosenblatt, M. (1979). Multivariate k-nearest neighbor density estimates. Journal of Multivariate Analysis, 9(1):1 – 15.

Macqueen, J. (1967). Some methods for classification and analysis of multivariate observations. In In 5-th Berkeley Symposium on Mathematical Statistics and Probability, pages 281–297.

Neumann, M. H. (1998). Strong approximation of density estimators from weakly dependent observations by density estimators from independent observations. The Annals of Statistics, 26(5):2014 – 2048.

Ooi, H. (2002). Density visualization and mode hunting using trees. Journal of Computational and Graphical Statistics, 11(2):328–347.

O'Brien, T. A., Kashinath, K., Cavanaugh, N. R., Collins, W. D., and O'Brien, J. P. (2016). A fast and objective multidimensional kernel density estimation method: fastkde. Computational Statistics & Data Analysis, 101:148–160.

Park, B. U. and Marron, J. S. (1990). Comparison of data-driven bandwidth selectors. Journal of the American Statistical Association, 85(409):66–72.

Pevný, T. (2016). Loda: Lightweight on-line detector of anomalies. Machine Learning, 102(2):275–304.

Polonik, W. (1999). Concentration and goodness-of-fit in higher dimensions: (asymptotically) distribution-free methods. The Annals of Statistics, 27(4):1210–1229.

Porikli, F. (2005). Integral histogram: a fast way to extract histograms in cartesian spaces. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), volume 1, pages 829–836 vol. 1.

Ram, P. and Gray, A. G. (2011). Density estimation trees. Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 627–635.

Ram, P., Lee, D., March, W., and Gray, A. (2009). Linear-time algorithms for pairwise statistical problems. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C., and Culotta, A., editors, Advances in Neural Information Processing Systems, volume 22. Curran Associates, Inc.

Rosenblatt, M. (1976). On the Maximal Deviation of k-Dimensional Density Estimates. The Annals of Probability, 4(6):1009 – 1015.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J. C., Smola, A. J., and Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. Neural Comput., 13(7):1443–1471.

Scott, D. W. (2015). Multivariate density estimation: theory, practice, and visualization. John Wiley & Sons.

Shafer, G. and Vovk, V. (2008). A tutorial on conformal prediction. Journal of Machine Learning Research, 9(12):371–421.

Shang, N. (1994). Tree-structured density estimation and dimensionality reduction. pages 172–176.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(8):888–905.

Shimazaki, H., S. S. (2010). Kernel bandwidth optimization in spike rate estimation. Journal of Computational Neuroscience, 29:171–182.

Sutton, C. D. (1994). Tree structured density estimation. *Computing Science and Statistics*, 26:167–171.

Tax, D. M. and Duin, R. P. (2004). Support vector data description. *Machine Learning*, 54(1):45–66.

Terrell, G. R. and Scott, D. W. (1992). Variable kernel density estimation. *Ann. Statist.*, 20(3):1236–1265.

Tsybakov, A. B. (2008). *Introduction to Nonparametric Estimation*. Springer Publishing Company, Incorporated, 1st edition.

Vovk, V. and Buntine, W. (2012). Conditional validity of inductive conformal predictors. In *In: JMLR: Workshop and Conference Proceedings*, pages 475–490.

Wang, B. and Wang, X. (2007). Bandwidth selection for weighted kernel density estimation.

Wasserman, L. (2006). *All of Nonparametric Statistics*. Springer-Verlag New York, Inc.

Wasserman, L., Ramdas, A., and Balakrishnan, S. (2020). Universal inference. *Proceedings of the National Academy of Sciences*, 117:201922664.

Wong, W. H. and Ma, L. (2010). Optional Pólya tree and Bayesian inference. *The Annals of Statistics*, 38(3):1433 – 1459.