Density Estimation Trees

Seminar Report: Database Systems

Department of Informatics - Database Technology
University of Zurich

Supervision
Prof. Dr. Michael Böhlen
Prof. Dr. Peter Widmayer
Arijit Khan

May 30, 2015
Contents

1 Introduction 1

2 Parzen-Window Density Estimation 3
   2.1 Definition 3

3 Density Estimation Tree 5
   3.1 Piecewise constant density estimate 5
   3.2 Loss Function 5
   3.3 Categorical and Ordinal features 7
   3.4 Learning algorithm 8
   3.5 Consistency 9

4 Comparison 11
   4.1 Adaptability within dimensions 11
   4.2 Adaptability between dimensions 11
   4.3 Importance of dimensions 12
   4.4 Outlier Detection 12
   4.5 Speed 13

5 Conclusion 14
Chapter 1

Introduction

In machine learning density estimation, an instance of unsupervised learning, is a fundamental task. Compared with supervised learning, this is a hard task, because it works without any ground truth regarding the estimated quantity. In supervised learning, decision trees have been widely used [1]. However, in this report I want to focus on decision trees for unsupervised learning. A recent paper [3] introduced Density Estimation Trees, a novel method to build Probability Density Functions from training samples in a histogram-like fashion. Density Estimation Methods can be roughly classified into parametric (e.g. Maximum Likelihood) vs nonparametric (e.g. Parzen Windows) models. Parametric Models try to fit a given distribution model to the training samples by tuning its parameter. To evaluate a new observation, only the fitted distribution function needs to be used. Nonparametric models however, generally don’t need to be initialized because they evaluate the density of an input by comparing it to the stored training samples, which can be computationally very expensive. Density Estimation Trees are nonparametrics models which need to be initialized before using, which makes them a special case in the nonparametric estimator domain. The learning algorithm partitions the space in a top-down manner into a tree structure and precaches a density for each partition. The partitions are found by minimizing a cost function through leave-one-out cross-validation (LOO-CV). To evaluate a new input, a Density Estimation Tree just returns the density stored in the partition the input belongs to, which makes it a very fast nonparametric estimator.

An interesting property of Density Estimation Trees is adaptability. They are adaptable between dimensions, which means that they don’t treat each dimension the same way. Some dimensions might have more influence on the density than others (i.e. noise sources). They are also adaptable within dimensions. This means that in regions with fast changes, smaller partitions are built than in regions with slow changes. Also, Density Estimation Trees are more interpretable than the usual nonparametric models. DETs can be used to detect clusters and outliers. By looking at how different dimensions are treated in the partitioning process i.e. training algorithm, the importance of different dimensions can be derived. It also provides simple sets of rules with which a certain part of the data (e.g. a partition)
may be found (e.g. on a database).

The remainder of this Report will focus on how DETs work and what properties they have that are difficult to find in other unsupervised learning algorithms. The first chapter will give a brief introduction to the Parzen Window estimator, a well known method for nonparametric estimation. The main part will be the next chapter which introduces DETs and specifies their details. After that I will compare DETs with Parzen Windows and reach to a conclusion.
Chapter 2

Parzen-Window Density Estimation

In this chapter I want to give an overview over the Parzen-Window Estimator. However, the focus of this report is to show the advantages of DETs over Parzen-Windows and therefore I will keep it brief.

In the early 1960s, Emanuel Parzen [2] invented a nonparametric Density Estimator called the Parzen-Window Estimator or sometimes also called KDE (Kernel Density Estimator). It has found widespread utility in areas like pattern recognition, classification, image processing etc. The method is in its core an interpolation technique.

Given an observation $x$, the Parzen-Window estimates the probability density by placing a window on $x$ and then gives a result based on a set of training samples by checking the percentage that fall inside the window. It also offers the possibility to use different kernels, which usually are themselves PDFs.

2.1 Definition

The Parzen-Window estimate is given by

$$P(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h_N^d} K \left( \frac{x - x_i}{h_N} \right)$$ (2.1)

where

- $N$: number of observations
- $h_N$: window width / bandwidth
- $d$: number of dimensions
- $x_i$: $i$th training sample
- $K(x)$: Kernel function

The parameter $h_N$ is usually chosen based on the number of available training samples $N$. The kernel function is usually a unimodal function and a PDF by itself, which makes it simple to guarantee that the estimated function $P(\cdot)$
satisfies the conditions to be a PDF. Typical kernels are the Gaussian and Uniform types.
Chapter 3

Density Estimation Tree

This chapter explains how density estimation can be done using decision trees based on [3]. First the estimator with its corresponding loss function will be defined. Another section will explain how continuous features can be mixed with nominal and ordinal ones. Finally a proof of correctness will be given.

3.1 Piecewise constant density estimate

The piecewise constant density estimate of a decision tree $T$ is given by:

$$\hat{f}_N(x) = \sum_{l \in \tilde{T}} \frac{|l|}{NV_l} I(x \in l)$$  \hspace{1cm} (3.1)

where

$T$: the density estimation tree  
$S$: Set of $N$ observations in $\mathbb{R}^d$  
$\tilde{T}$: set of leaves of $T$ representing the partitions of the data space  
$|l|$: number of observations of $S$ in the leaf $l$  
$V_l$: volume of leaf $l$ within $d$ dimensional bounding box of $S$  
$I$: indicator function

The space is partitioned into pieces represented by different nodes of our tree. The density estimate is nothing more than the number of training samples (i.e. observations) encountered in all leaves containing the evaluated position, divided by the total number of training samples and the volume of the partitions.

3.2 Loss Function

Unsupervised learning of a density estimation tree in the above form requires a notion of a loss function that is minimized using a greedy learning algorithm. In [3] the Integrated Squared Error (ISE) loss function [4] was used. In comparison
3.2. Loss Function

with maximum-likelihood based functions, the ISE gives a robust notion of the error between the estimated and true density.

To learn a DET we have to solve the following optimization problem:

$$\min_{f_N \in \mathcal{F}_N} \int_X \left( \hat{f}_N(x) - f(x) \right)^2 dx$$

(3.2)

where $\mathcal{F}_N$ is the set of all the estimators in the form of Eq. 3.1 that can be learned using a set of $N$ training samples. Now we expand the square:

$$\min_{f_N \in \mathcal{F}_N} \int_X \left( \hat{f}_N(x)^2 - 2 \hat{f}_N(x)f(x) - f(x)^2 \right) dx$$

the term $f(x)$ is constant among different estimators $\hat{f}_N(x)$ and thus $f(x)^2$ doesn’t have to be considered during this optimization problem. By using the following Monte-Carlo Substitution

$$\int_X \hat{f}_N(x)f(x) dx \approx \frac{1}{N} \sum_{i=1}^N \hat{f}_N(X_i),$$

where $X_i$ refers to the $i$’th training sample, we obtain the following estimator of the ISE [4]:

$$\min_{f_N \in \mathcal{F}_N} \left\{ \int_X \left( \hat{f}_N(x) \right)^2 dx - \frac{2}{N} \sum_{i=1}^N \hat{f}_N(X_i) \right\}$$

(3.3)

Now we plug in the piecewise constant density estimator from Eq. 3.1. Substituting

$$\hat{f}_N^2(x) = \sum_{l \in \tilde{T}} \left( \frac{|l|}{NV_l} \right)^2 \mathbb{1}(x \in l)$$

(the cross terms in the expansion of $\hat{f}_N^2(x)$ vanish because of the indicator function) and

$$\sum_{i=1}^N \hat{f}_N(X_i) = \sum_{i=1}^N \sum_{l \in \tilde{T}} \frac{|l|}{NV_l} \cdot \mathbb{1}(X_i \in l) = \sum_{l \in \tilde{T}} \frac{|l|}{NV_l} \cdot |l|$$

we get the following objective function

$$\int \left\{ \sum_{l \in \tilde{T}} \left( \frac{|l|}{NV_l} \right)^2 dx - \frac{2}{N} \sum_{l \in \tilde{T}} \frac{|l|}{NV_l} \cdot |l| \right\}$$

(3.4)

Doing another substitution,

$$\int \left( \frac{|l|}{NV_l} \right)^2 dx = \left( \frac{|l|}{NV_l} \right)^2 \int dx = \left( \frac{|l|}{NV_l} \right)^2 \cdot V_l$$
the estimator of the ISE for the DET turns into the following equation:

\[
\sum_{l \in \tilde{T}} \left\{ \left( \frac{|l|}{NV_l} \right)^2 \cdot V_l - 2 \left( \frac{|l|}{NV_l} \cdot |l| \right) \right\} = \\
\Rightarrow \sum_{l \in \tilde{T}} \left\{ \frac{|l|^2}{N^2V_l} - 2 \cdot \frac{|l|^2}{N^2V_l} \right\} = \\
\Rightarrow \sum_{l \in \tilde{T}} \left\{ - \frac{|l|^2}{N^2V_l} \right\}
\]

Thus, the greedy surrogate of the error for any node \( t \) is denoted by

\[
R(t) = - \frac{|t|^2}{N^2V_t}
\]

(3.6)

The tree is then grown in a top down manner by minimizing this greedy surrogate of the error over the available training samples.

### 3.3 Categorical and Ordinal features

In order to do density estimation over data with mixed (i.e. continuous, categorical and ordinal) features, [3] defines a novel density estimator and a loss function that allow mixing of different feature types and may be used to learn DET’s:

The piecewise constant density estimate of a decision tree \( T \) with mixed features is given by:

\[
\hat{f}_N(x) = \sum_{l \in \tilde{T}} \frac{|l||I(x \in l)|}{N \cdot V_{l_d} \cdot \prod_{j=1}^{d'_l} R_{t_j} \cdot \prod_{i=1}^{d''_l} M_{t_i}}
\]

(3.7)

where

- \( T \): the density estimation tree
- \( S \): Set of \( N \) observations in \( \mathbb{R}^d \times \mathbb{Z}^{d'} \times \mathbb{C}^{d''} \)
- \( d \): number of real features
- \( d'_l \): number of ordinal features
- \( d''_l \): number of categorical features
- \( \tilde{T} \): set of leaves of \( T \) representing the partitions of the data space
- \(|l|\): number of observations of \( S \) in the leaf \( l \)
- \( V_{l_d} \): volume of leaf \( l \) within \( d \) dimensional bounding box of \( S \)
- \( R_{t_j} \): range of ordinal values in the \( j^{th} \) of the \( d'_l \) ordinal dimensions present in \( l \)
- \( M_{t_i} \): number of categories present in \( l \) for the \( i^{th} \) of the \( d''_l \) categorical dimensions present in \( l \)
- \( I \): indicator function

Similar to the continuous features case, we can define the greedy surrogate of the error for any node \( t \) as:

\[
R(t) = - \frac{|t|^2}{N^2 \cdot V_{l_d} \cdot \prod_{j=1}^{d'_l} R_{t_j} \cdot \prod_{i=1}^{d''_l} M_{t_i}}
\]

(3.8)
3.4 Learning algorithm

In order to find an optimal tree \( T \) the CART (Classification and Regression Tree) model \[1\] is used in \[3\]. However, since we’re doing unsupervised learning, a different optimization function given by Eq. 3.8 is used.

In general, the idea is the following:

1. Grow a large binary tree using forward selection. For each node, find the best split among all possible splits and dimensions. Grow until are leaf nodes either:
   - have \( < n \) data points, where \( n \) is a predefined parameter
   - have a sufficiently small cost given by Eq. 3.8
2. Prune the tree back, creating nested sequence of trees, decreasing in complexity.

Splitting: For each node, the tree needs to be split so that the summed up loss of the nodes is minimized. We denote a split by \( s \in S \) where \( S \) is the set of all possible splits and the best split by:

\[
s^* = \arg \max_{s \in S} \{R(t) - R(t_L) - R(t_R)\}
\] (3.9)

where the children of the node \( t \) satisfy \( |t| = |t_L| + |t_R| \). For continuous and ordinal dimensions, the optimal splits are found by trying every \( |t| - 1 \) possible splits. For categorical dimensions, each combination of two subsets of the categories available at the nodes are evaluated, which gives \( 2^{k-1} - 1 \) possible splits.

Pruning: Overfitting is a problem often encountered in machine learning algorithms. To avoid this \[3\] introduces a regularization term, which reduces the model complexity, and adds it to the greedy surrogate error

\[
R_\alpha(t) = R(\tilde{t}) + \alpha \cdot |\tilde{t}|
\] (3.10)

where \( \alpha \) is a regularization parameter found by cross-validation and \( \tilde{t} \) is the set of leaves in the subtree rooted at \( t \). During the training process of \( T \) \( \alpha \) is gradually increased. A subtree that is rooted at \( t \) is pruned for the value of \( \alpha \) where \( R_\alpha(t) = R_\alpha(\{t\}) \) is the regularized error of the pruned subtree. The search space of \( \alpha \) is finite and therefore it is efficient to calculate \( \alpha \) efficiently \[1\].

Cross Validation: Cross Validation is a technique often encountered in machine learning approaches. It is used to find the optimal value for parameters governing the training behaviour, like in this case \( \alpha \). From \[4\] we obtain the following expression for our LOO-CV estimator:

\[
\hat{J}(\alpha) = \int_X \left( \hat{f}_N^\alpha(x) \right)^2 dx - \frac{2}{N} \sum_{i=1}^N \hat{f}_{(-i)}^\alpha(X_i)
\] (3.11)
\( \hat{f}_N \) denotes the estimator with the decision tree \( T \) pruned with parameter \( \alpha \), whereas \( \hat{f}_{\alpha - i} \) is the estimator with the decision tree \( T_{\alpha - i} \) and sample \( i \) excluded from the training set. The tree with optimal size is the tree \( T \) with parameter \( \alpha^\ast \) such that:

\[
\alpha^\ast = \arg\min_{\alpha} \hat{J}(\alpha)
\] (3.12)

### 3.5 Consistency

How do we know, if an estimator converges towards a useful result during training? We can find that out by proving its consistency. A consistent estimator is an estimator whose output converges towards the true values if the number of training samples converges towards infinity. The variance of the error between true and estimated value goes towards zero. Mathematically, convergence of a nonparametric estimator can be proven by showing that [2]:

\[
\Pr\left( \lim_{N \to \infty} \sup_{t \in \mathcal{B}} |\hat{F}_N(t) - \int_t f(x) \, dx| = 0 \right) = 1
\] (3.13)

In this report I will only give the proof for continuous functions based on [3]:

**Theorem 1.** The estimator \( f_N \) from Eq. 3.7 satisfies Eq. 3.13.

**Proof.** Let \( \mathcal{B} \) be the collection of all sets \( t \subset X \) that can be interpreted as the solution set to a system of \( k \) inequalities of the form \( b^T x \leq c \) where \( b \in \mathbb{R}^d \) and \( c \in \mathbb{R} \). Each leaf \( l \in \tilde{T} \) in a decision Tree \( T \), is the solution set of a system with \( k \) inequalities of the form \( b^T x \leq c \) with \( b \in \mathbb{R}^d \) and \( b \) has exactly one entry equal to 1 and the others equal to 0. Hence, the partitions that are generated in a Density Estimation Tree are a subset of all possible solutions to a system of inequalities in the above form, i.e. \( \tilde{T} \subset \mathcal{B} \).

Let \( X_i \) where \( i \geq 1 \) be a random sample from a pdf \( f \) on \( X \). For \( N \geq 1 \), let \( \hat{F}_N \) denote the empirical distribution of \( X_i \) where \( 1 \leq i \leq N \), defined on a set \( t \subset X \) by

\[
\hat{F}_N(t) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}(X_i \in t) = \frac{|t|}{N} = \int_t \hat{f}_N(x) \, dx
\] (3.14)

where \(|t|\) is the number of random samples in the set \( t \cap \{x_i|1 \leq i \leq N\} \) which are all the training samples that fulfill a set of inequalities imposed by the node \( t \) i.e. fall into the corresponding partition of the sample space and \( \hat{f}_N(x) \) is the estimator given in Eq. 3.1.

According to a general version of the Glivenko-Cantelli theorem [5], the following holds:

\[
P\left( \lim_{N \to \infty} \sup_{t \in \mathcal{B}} |\hat{F}_N(t) - \int_t f(x) \, dx| = 0 \right) = 1
\] (3.15)

To put it simple, this means that the probability that the largest possible error between the estimated and the real distribution of any possible partition \( t \) is
equal to 0 converges to 1 if we increase the sample size towards infinity. By substituting the left term in the supremum by Eq. 3.14 we get:

\[
P \left( \lim_{N \to \infty} \sup_{t \in B} |\hat{f}(x)dx - \int f(x)dx| = 0 \right) = 1
\]

\[
\Rightarrow P \left( \lim_{N \to \infty} \sup_{t \in B} \int |\hat{f}(x)dx - f(x)dx| = 0 \right) = 1
\]

(3.16)

Now we make the assumption that if the sample size converges towards infinity, the partition sizes get infinitesimally small, i.e. \(\lim_{N \to \infty} P(\text{diameter}(t) \geq \epsilon) = 0\), because each leaf can only have a bounded number of points. We come to the following conclusion with probability 1:

\[
\lim_{N \to \infty} \sup_{t \in B} \inf_{t} |\hat{f}_N(x) - f(x)|dx \leq \lim_{N \to \infty} |\hat{f}_N(x') - f(x')| \cdot \int dx = 0
\]

for some \(x' \in t\). Therefore

\[
P \left( \lim_{N \to \infty} \sup_{t \in B} \int |\hat{f}_N(x) - f(x)|dx = 0 \right) = 1 \Rightarrow P \left( \lim_{N \to \infty} \int_X \left( \hat{f}_N(x) - f(x) \right)^2 dx = 0 \right) = 1
\]

\(\square\)
Chapter 4

Comparison

To measure the accuracy of DETs compared to KDEs and histograms, in [3] they did different experiments.

4.1 Adaptability within dimensions

The first experiment was to fit a DET, Histogram, KDE and a local r-KDE (rodeo KDE, special version of Parzen Window Estimator) to a sample set of 1000 points. To have any ground truth to compare it to, an artificial sample set was used generated by a strongly skewed distribution given by

\[ X \sim \sum_{i=0}^{7} \frac{1}{8} \mathcal{N} \left( 3 \left( \left( \frac{2}{3} \right)^i - 1 \right), \left( \frac{2}{3} \right)^{2i} \right). \]

The results can be seen in Fig. 4.1. The first thing to notice, is that the DET has a smooth tail, whereas the other methods have wiggly tails due to a fixed bandwidth. As mentioned before, one main advantage of DETs is the adaptable bandwidth within dimensions. It can be seen that in areas with fast changes around the peak, the partitions are very small whereas in areas with slow changes above 0 the partitions get large. The other methods have to choose small windows, because otherwise they couldn’t fit the peak which makes the tail look more noisy. If on the other hand one would choose larger windows, the curves would become smoother but the peak wouldn’t show.

4.2 Adaptability between dimensions

In the second experiment two dimensional data was used. In one dimensions there a mixture of beta distributions and in the other a simple uniform
4.3 Importance of dimensions

Another intuitive feature of DETs is that the importance of dimensions can easily be read from the tree. Dimensions with splits high up near the root node have a greater impact on the density than dimensions that are split further down. Additionally, dimensions that don’t contribute to density at all are never split as can be seen in Fig. 4.2.

4.4 Outlier Detection

What happens when we have outliers in our data set? Outliers can roughly be defined as points lying far off clusters. If we run the DET algorithm on such a dataset, we will get large partitions with very low density in outlier regions. Therefore, to find outliers, we only have to look at large leaf nodes, with a low density.
4.5 Speed

In [3] they compared the DETs to KDEs using the fast approximate dual-tree algorithm with kd-trees. To estimate the optimal bandwidth for the KDEs they used LOO-CV which is the reason why the KDEs have a training time. The results showed, that the training time 2 to 4 times higher when using DETs. However, the query time is many orders of magnitude lower for DETs.
Chapter 5

Conclusion

This framework of Density Estimation Trees has some significant advantages over other nonparametric density estimators. They are defined by a simple set of rules which are intuitive and easily interpretable. It is possible to not only model continuous, but also ordinal and categorical dimensions. Also they perform feature selection, a property seldom found among other nonparametric density estimators.

New points can be evaluated cheaply like in regression and classification trees due to the simple tree structure. However, these advantages come with a cost. One problem that remains are discontinuities along the borders of different partitions. However, this problem could be solved for instance by smoothly interpolating the density values along different partitions using for instance MLPs (multilayer perceptrons) or other such methods.

Nevertheless, DETs can be applied to different kinds of data analysis like outlier and anomaly detection. They could also be used in embedded systems due to their computational efficiency.
Bibliography


