Department of Computer Science

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Hamilton, New Zealand

Tree-based Density Estimation: Algorithms and Applications

by

Gabi Schmidberger

This thesis is submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy in Computer Science
at The University of Waikato

February 2009

Abstract

Data Mining can be seen as an extension to statistics. It comprises the preparation of data and the process of gathering new knowledge from it. The extraction of new knowledge is supported by various machine learning methods. Many of the algorithms are based on probabilistic principles or use density estimations for their computations. Density estimation has been practised in the field of statistics for several centuries. In the simplest case, a histogram estimator, like the simple equal-width histogram, can be used for this task and has been shown to be a practical tool to represent the distribution of data visually and for computation. Like other nonparametric approaches, it can provide a flexible solution. However, flexibility in existing approaches is generally restricted because the size of the bins is fixed—either the width of the bins or the number of values in them. Attempts have been made to generate histograms with a variable bin width and a variable number of values per interval, but the computational approaches in these methods have proven too difficult and too slow even with modern computer technology.

In this thesis new flexible histogram estimation methods are developed and tested as part of various machine learning tasks, namely discretization, naive Bayes classification, clustering and multiple-instance learning. Not only are the new density estimation methods applied to machine learning tasks, they also borrow design principles from algorithms that are ubiquitous in artificial intelligence: divide-and-conquer methods are a well known way to tackle large problems by dividing them into small subproblems. Decision trees, used for machine learning classification, successfully apply this approach. This thesis presents algorithms that build density estimators using a binary split tree to cut a range of values into subranges of varying length. No class values are required for this splitting process, making it an unsupervised method. The result is a histogram estimator that adapts well even to complex density functions—a novel density estimation method with flexible density estimation ability and good computational behaviour.

Algorithms are presented for both univariate and multivariate data. The univariate histogram estimator is applied to discretization for density estimation and also used as density estimator inside a naive Bayes classifier. The multivariate histogram, used as the basis for a clustering method, is applied to improve the runtime behaviour of a well-known algorithm for multiple-instance classification. Performance in these applications is evaluated by comparing the new approaches with existing methods.

Acknowledgments

Sincerest thanks and appreciation to my supervisors, Eibe Frank, Geoff Holmes and Mark Hall for their support and guidance throughout my work on this thesis.

I would also like to thank Grant, Sven and Claire for the occasional discussion of one or the other point of research and Peter for his continuous readiness to help out with whatever technical problem had arisen.

Further thanks to Christine and Wolfgang for their wise advice on organisational matters, my friends from *grrlcamp* and *now/here* for their invaluable moral support and all fellow graduate students for being my friends at Waikato University despite our large age difference.

Most of all thanks to my husband Bernhard and children Boris and Barbara for enduring a long string of '20-minute' or take-away meals and my children especially for never forgetting to ask: "...and when will you be finished with your thesis, Gabi?"

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine learning has grown into a mature field of research. In the research literature and at conferences a great number of machine learning methods and algorithms have been developed and discussed. The "Machine Learning" journal [42] was first published in March 1986. This is now more than 20 years ago. Also the "International Conference on Machine Learning (ICML)" just recently celebrated its 25^{th} anniversary.

Machine learning's goal is to find a way to make computers gain 'knowledge' or improve their performance from experience [45]. Tom Mitchell calls machine learning a natural outgrowth of the intersection of computer science and statistics [45]. Statistics, a much older discipline, also aims at inferring facts from data. The first statistical techniques were developed hundreds of years ago and statistics has been widely practised in the pre-computer age. Nowadays, both computer scientists and statisticians work in the field of machine learning. The book "Elements of Statistical Learning" [26], published in 2001, summarizes machine learning methods from the statisticians' point of view.

Artificial intelligence (AI) is an area of research that aims to mimic functions of human intelligence using computers. Machine learning is a subfield that has its roots in artificial intelligence. Its objective is to enable a computer, or more general a machine, to learn [45]. Other prominent topics in artificial intelligence are *planning* and *knowledge representation* to name a couple. AI topics provided the incentive to develop a range of algorithms, e.g. search algorithms. The size of AI tasks often makes it necessary to develop the algorithms in a way that uses computation time wisely. Tom Mitchell points out that the field machine learning always had to take into consideration what is computationally feasible [45].

Outside the research communities the technical term *data mining* is perhaps better known than the term machine learning. Data mining generally refers to the process of gaining knowledge from data in a commercial environment and addresses the full knowledge extraction process, starting from preparing the data, to exploring the data and finally gaining knowledge from it using machine learning methods. Data mining mostly involves working with large volumes of data. This itself makes the efficiency of algorithms very important. Note that prominent researchers (Han & Kamber [24]) believe that data mining is actually not the best name for the task. Like it is not the sand that is mined but the gold, it is not data that is mined but the knowledge from the data.

Some of the main machine learning tasks are: *classification*, which aims to build a model to use for prediction of the label (i.e. class value) of later unseen data instances; *regression*, which does the same as classification but for a numeric target value instead of a categorical class value; *clustering*, which aims to identify areas of high density in the data that can potentially be used to define labels for the instances; and *association rule mining*, which searches for rare but important patterns in the data.

These machine learning tasks have been supported with various different techniques, some of them using density estimation to find a good model. There is a strong connection between density estimation, clustering and classification. Clustering is looking for dense areas surrounded by less dense areas. Density estimation can help finding these dense areas. Classification also selects areas where specific class values are predominately found, i.e. where the density of examples with these values is high. In both applications it is important to find good thresholds to decide where the area of a cluster or the area pertaining to one class begins and where it ends. Density models which build a density function by setting thresholds are binning methods like the simple histogram and are also called discretization methods. In this thesis new binning algorithms for density estimation are developed and applied to some of the above-mentioned machine learning tasks.

Density estimation originates in statistics. The so-called statistical frequentists are the followers of Fisher [3], who—among other things—pioneered *parametric density estimation*. Parametric density estimation requires the assumption of a distribution family (e.g. Gaussian) and then the given dataset is used to estimate the parameters of the distribution. In contrast, nonparametric approaches like Parzen's [48] do not assume a specific functional form

of the distribution. The major difference of *nonparametric density estimation* is that for each value of the estimated density function only instances within a certain distance from the query point are taken into account. The width of this 'window' is crucial for accurate estimation. Nonparametric density estimation is able to model more complex density functions than standard parametric methods [55].

A first step in data analysis is 'looking at the data' that equates to the *data exploration* step of the modern data mining process. The data exploration step was inspired by Tukey [27] who developed *exploratory data analysis* as part of a movement away from frequentist statistical thinking. In this approach the expert inspects the data using data visualization tools. One of the most prominent data visualization tools, namely the histogram, was in fact, invented some hundred years ago. A histogram can not only be used to explore the data, but, if used as a model for classification or clustering, can also provide an easily understandable model of the data.

One of the earliest successful classification techniques is decision tree classification [9][50]. These classifiers analyze a given data sample, consisting of a set of instances which are each described by a fixed set of attributes (or features), and build a tree-based model. At each node of this tree the classifier decides on branching further down the tree depending on the values of one of the attributes of the instances concerned. As soon as a leaf node is reached, the class is defined, using the majority class of training instances in this leaf. Decision tree classifiers can be constructed in an efficient way using a so-called 'greedy' algorithm to build the tree recursively from the root node down. This way, a partition on the example data is defined using a locally best solution—without testing all possible partitions—but this still generally results in a reasonably good partition. In most cases testing all possible solutions is not computationally feasible.

In this thesis, tree-based algorithms for density estimation are developed. The algorithms use a tree-based method to split ranges of univariate and multivariate data into subranges such that the subranges represent uniform distributions. As in tree learning for classification, a greedy method is used for the selection of split points. Then, on the basis of the resulting non-overlapping subranges, a density histogram can be formed. These histogram estimators are applied to various machine learning tasks and evaluated in the context of these tasks. Visualizing tools—like histograms for univariate data and multivariate summarization tools for multivariate data—developed in this thesis, help to

explain the results. Experimented evaluation was done using the WEKA machine learning tool set [68].

The next subsections provide a brief introduction to the basic concepts used throughout this thesis. Section 1.1 explains the main principles of density estimation. A second basic concepts section, Section 1.2, establishes the machine learning terms used in this thesis. Both these sections prepare for the motivation for this thesis presented in Section 1.3. Finally, Section 1.4 explains the overall thesis structure.

1.1 Density Estimation

The work in this thesis is on density estimation for numeric data. Density estimation is the process of constructing an estimate of the density function underlying the given example (or sample) data. Following standard machine learning terminology, the example data is also called the training data for the density model.

Density estimation has been practised in statistics and probability theory before data mining was even known. At the beginning of the 20^{th} century, Fisher introduced his parametric methods of density estimation [3] which are still widely practised. These methods require the statistician to choose a model from a possible set of distributions of the data (e.g Gaussian) and use the example dataset to adapt the parameters of this model, thus forming an estimate of the true unknown distribution.

In contrast, nonparametric methods for density estimation are a more recent approach and partly rival the parametric methods. A much older method, the density histogram is also a nonparametric density estimator. The simplest type of density histogram is built by splitting the range of values into intervals of equal length. The number of intervals (i.e. bins) is given by the user. It is quite common to explore new data by visualizing it with equal-width histograms and varying the number of bins.

In histograms, the number of bins fulfils the role of the so-called *smoothing factor*, which in some form plays a role in most nonparametric density estimation techniques. If a small number of bins is chosen, it results in most cases in a smoother distribution function estimate when compared to a larger number of bins. A histogram that does not show enough details of the data distribution is called *over-smoothed*. A histogram that shows too much detail is called *under-smoothed*. Figure 1.1 gives an example of two histograms of data from

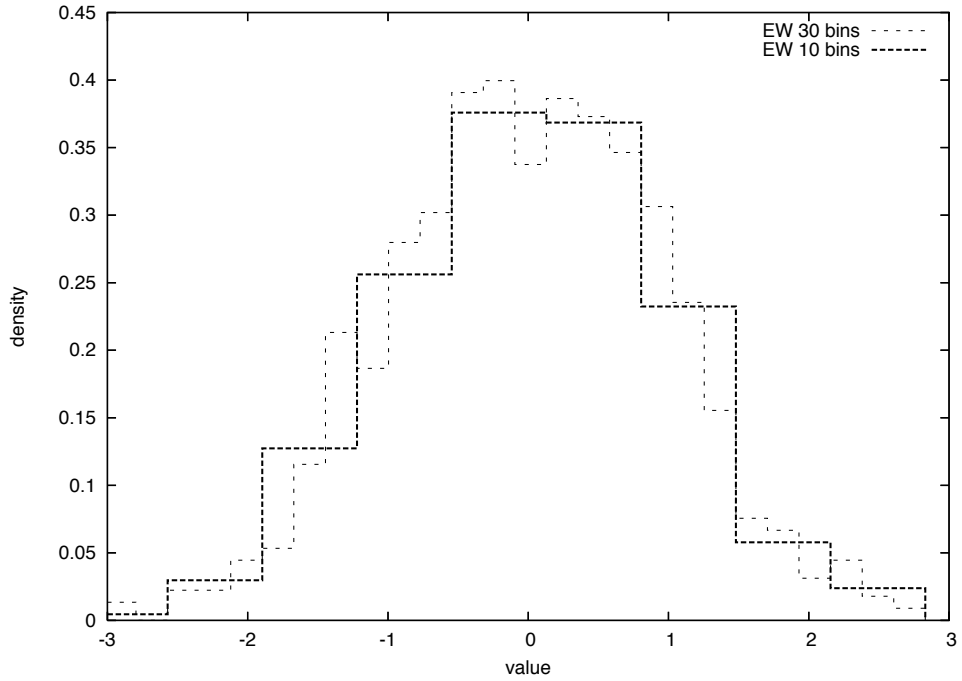

Figure 1.1: Example of splitting an unknown distribution.

a simple unknown distribution function, two equal-width histograms with 10 and 30 bins respectively. In the second histogram—the histogram with the higher number of bins—two maxima (i.e. modes) are visible. A question that arises is whether the second histogram is under-smoothed or the first one over-smoothed.

For data from a simple Gaussian distribution, Scott [55] developed a formula to compute an appropriate number of bins for a histogram estimator. A more general method to find this number is to cross-validate for the number of intervals. The cross-validation method tests a model using the example data as training and test data. It averages a test criterion between n versions of the model. To build these n models the sample data is split into n parts and, for each test, $n-1$ parts are taken as training data to build the model and one part is taken as test data. Cross-validation is explained in more detail in Chapter 2.

1.2 Basic Concepts

Most of the experiments in this thesis work with the standard data format that is commonly used in machine learning. The data is organized in a dataset: a set of instances where each instance corresponds to an example (i.e. observation)

from a real-world or artificial data source. Each instance consists of a fixed-length attribute vector. Each attribute (or feature) of an instance describes an aspect of the example. Some machine learning algorithms assume one of the attributes to be special and assume it gives the label or class of the example. Methods using a class attribute are *supervised* methods. *Unsupervised* methods do not use a class attribute.

The most common attribute types are the categorical and the numeric attribute type. Numeric attributes have continuous (i.e. real) or discrete values (e.g. integer values). Categorical attributes have a set of labels as possible values. 'Wellington', 'Auckland', 'Christchurch', 'Hamilton' and 'Dunedin' are examples of labels of a categorical attribute containing the names of towns in New Zealand. Categorical data types are normally unordered like this, but can also be ordered. For example the attribute 'fever' could have the ordered values 'below normal temperature', 'normal temperature', 'light fever', 'high fever'.

There can be slight differences in the definition of attribute types depending on the tool used. In the WEKA mining tool set [68], numeric attributes can be both discrete and continuous. Like most machine learning tools it can also represent values as missing. Note that the algorithms developed in this thesis assume numeric data without missing values.

A more complex data structure are multiple-instance datasets where examples contain multiple instances rather than one. The multiple-instance data format is used in an application of the multivariate algorithm developed in this thesis and is explained in detail in Section 6.2.

The basic machine learning tasks relevant to this thesis are classification and clustering. Classifiers are tools for the prediction of a categorical class value. This means they use the class value for the construction of a model and therefore are supervised. A classification method takes training data—which in this case is data with known class values—and builds an algorithm-specific model to predict the class attribute of further, so far unseen instances. Classifiers are induction methods because they conclude from the specific to the general. The model of the classifier can be seen as a function. The input parameters of this function are the attribute values and the output value is the class value of the instance.

Clustering looks for instances with high similarities that form a group distinct from all other instances in the dataset. Clustering starts with data that does not have a label yet, which means it is an unsupervised method. Clus-

tering is done to find new patterns and can also be used for data exploration to gain insight into the structure of the data. It looks for multidimensional modes, i.e. areas of high density which are surrounded by areas of lower density. These areas are the clusters and define the model. The resulting clusters can be used to define new concepts, i.e. classes, on the data.

In classification and clustering the model building step is also called *training*. As in statistics, machine learning tries to infer new facts from the given data, under certain modelling assumptions. It is important to remember that these modelling assumptions place a bias on the results. This bias can be strong, as in parametric methods, or weak, as in nonparametric ones.

Classifiers are evaluated by measuring their prediction accuracy. For testing, the training dataset or a different evaluation dataset (i.e test dataset) is taken, and the class values of the examples in this set are predicted using the model and then the real class values are compared with the predicted ones. The percentage of correctly predicted instances is called the accuracy of the classifier. Note that the evaluation using the training data, with which the model was also built, generally gives a very optimistic estimate of accuracy.

In contrast to classifiers, clustering results are in general difficult to evaluate because it is rare that a test dataset is provided with correct cluster assignments.

Further, more specific background information about methods used in this thesis, e.g. tree-based algorithms, is given in Chapter 2.

1.3 Motivation

Density estimation is a frequently used tool in machine learning, as it is for instance used in clustering and in probability-based classification. Histogram estimators are nonparametric density estimators and are able to represent complex density functions. However, existing methods which are in common use are restricted by either a fixed bin width or a fixed number of instances per bin. The number of bins is commonly given by the user. Existing methods for histograms with varying bins (width or number of instances) are too computationally intensive to be of practical use [56]. Moreover, existing techniques, which are in common use, are only univariate.

Ideally, histogram techniques should produce an estimator that is simple, yet represents the distribution well. This thesis develops new algorithms that generate histogram estimates with variable bins for numeric univariate and

multivariate data. The evaluation of these algorithms aims to answer the following questions (sections with corresponding empirical results are given in brackets):

- Can greedy algorithms for tree-based density estimation represent the structure of the distribution function well, by adapting to all significant changes in the density function, and also abrupt changes and areas of (effectively) zero density? (4.1) For instance, can tree-based density estimation be used to find clusters in multidimensional data? (6.1)

- Can the algorithms employ cross-validation to determine an appropriate number of bins based on the input data alone? (4.1)

- Can the induced nonparametric density estimators support density estimation tasks in machine learning applications, such as single-instance and multiple-instance classification, so that the performance in the target application is improved when the new estimator is used to augment an existing approach or used in place of standard histogram estimators? (4.2, 6.2)

- Can tree-based density estimators generate density models based on computational requirements that render them useful in practical applications? (4.1, 4.2, 6.1, 6.2)

The thesis also investigates—albeit in a less formal manner—whether interpreting data using visualization based on the histograms produced can potentially aid exploratory analysis. However, the focus is on the quantitative evaluation of algorithms for tree-based density estimation.

1.4 Thesis Structure

The content of this thesis is organized in seven chapters, including this introduction, which provides context and motivation, and the conclusion chapter, which summarizes the contributions.

Chapter 2 introduces the underlying concepts necessary for constructing tree-based density estimators in more detail. It is split into two parts: the first part explains density estimation and corresponding techniques developed in statistics; the second part covers tree construction, a topic stemming primarily from machine learning. The discussion of density estimation is focused

on histogram estimators. Tree construction has been widely used for decision tree classification in machine learning and is a classic divide-and-conquer algorithm—a technique often used in search problems.

The next two chapters, Chapters 3 and 4, cover the univariate version of the tree-based density estimation method developed in this thesis. Chapter 3 explains in detail the algorithm for the first tree-based density estimation method introduced in this thesis, called *TUBE*. TUBE stands for Tree-Based Unsupervised Bin Estimator. A major part of the algorithm involves finding the split points between the bins and deciding where to cut exactly. The splits are selected in a greedy fashion and the process is repeated recursively on the subranges produced. In addition to the splitting criterion, which decides where to split, a stopping criterion is defined to decide when to stop splitting and thus implicitly select an appropriate number of bins.

Chapter 4 investigates two applications of univariate density estimation trees. In the machine learning context, the process of splitting a numeric attribute into subranges is called discretization. In the first application, TUBE's univariate discretization is evaluated as a density estimator using the log-likelihood criterion. In the second application, TUBE discretization is used inside the classification algorithm naive Bayes and is compared with other univariate methods for nonparametric density estimation in this context. For the second application, the evaluation criterion is classification accuracy.

Chapters 5 and 6 cover the multivariate version of the tree-based density estimation method developed in this thesis. Chapter 5 contains the implementation details of the multivariate version of the algorithm, called *Multi-TUBE*. Most parts of Chapter 3, which introduces the univariate version of the algorithm, are also relevant for the multivariate case because Multi-TUBE can be seen as a generalization of the univariate density estimation algorithm. In Multi-TUBE, splits are performed on any of the attributes in the data, instead of a single attribute as in the univariate case. Obviously visualization of multivariate histograms with more than three attributes is not possible in a standard fashion. Chapter 5 also introduces some simple methods to list a histogram's bins in a user-friendly way, to give some insight into the structure of a multidimensional histogram.

Chapter 6 covers two applications of multivariate density estimation trees. The first application investigates the possibilities of using the multivariate tree-based density estimation tool for clustering. The second application uses the clustered bins of the multivariate histograms for classification of multiple-

instance problems. These problems exhibit the multiple-instance data format, where an example consists of a bag of unlabelled feature vectors and the class value is assigned to this bag of instances.

As already indicated above, the major contributions of this thesis are listed in Chapter 7, which also summarises the results of this thesis and suggests future work.

Chapter 2

Density Estimation and Tree Construction

The density estimation methods developed in this thesis use a binary tree to construct a density estimation function. They are adaptations of an algorithm widely used in machine learning to the statistical task of density estimation. This chapter discusses the two main underlying concepts, density estimation and tree construction. Further content in this chapter covers related work, such as similar algorithmic approaches to density estimation.

Density estimation is the task of estimating the underlying density function of data on the basis of a sample from this data. Density estimation techniques are often split into two groups: parametric density estimation techniques and nonparametric ones. Parametric methods assume a known distribution and the sample data is used to compute the parameters for this distribution. Nonparametric methods do not assume a particular underlying distribution function. For example, histograms are considered to be a nonparametric density estimation method. The density estimation methods in this thesis do not assume an underlying distribution and generate histograms—one-dimensional and multi-dimensional ones. Hence, like the standard histogram method, they can be seen as nonparametric methods. Note that another nonparametric approach is Kernel density estimation [30][48], which has grown into a very large research area in the last few years. This thesis does not cover kernel estimation methods.

In this thesis 'constructing density estimation trees' means using a tree building algorithm to construct a histogram. The subranges of the histogram bins are found in a divide-and-conquer fashion by recursively splitting the range and building a binary split tree.

The content of this chapter is organized in the following way. Section 2.1 gives an overview of density estimation with a detailed explanation of existing histogram techniques. Section 2.2 explains the principles of tree construction as it is used in other areas of machine learning. Section 2.3 introduces further related methods.

2.1 Density Estimation

For the statistical concept known as *probability density function* the values of a dataset are considered to be of a random variable X. The density function f describes the distribution of this variable X. It provides probabilities for the values of the dataset based on the following relationship (see also [60]):

$$P(a < X < b) = \int_a^b f(x)dx \qquad for\ all\ \ a < b \tag{2.1}$$

Density estimation estimates this probability density function using a sample of data points from the domain (i.e. data source). It is assumed that the sample represents the underlying target domain sufficiently well. Density estimation techniques differ in their tolerance to flawed data.

2.1.1 Parametric and Nonparametric Estimation

As stated above, density estimation methods can be classified as parametric or nonparametric methods. For parametric methods, a type of function, for instance the Gaussian density function, is assumed, and the density estimation method finds the parameters of this distribution function to fit the given sample data. The estimation of these parameters can be computationally expensive. The more dimensions the dataset has, the more parameters are involved and thus have to be computed. Another significant problem is that the data may not follow the chosen type of distribution and it is generally impossible to know the type of distribution for a high-dimensional dataset.

There is some disagreement as to when an estimator is nonparametric. In fact, Scott [56] states that it is difficult to define exactly what distinguishes a nonparametric estimator from a parametric one. He mentions that there is a notion to define a nonparametric estimator by the fact that the number of its parameters depends on the size of the sample or even is infinite. A more intuitive definition of nonparametric estimation is given in Terrell and Scott [63]. This work defines a nonparametric estimator as asymptotically

local. Local means that only local instances have influence on the point density estimate $f(x)$ (the density value at a point x). Parametric density estimators are not local.

Because they are locally defined, nonparametric methods can adapt better to complex distributions and therefore they are more flexible. Of course, if they are extremely local, like for example a histogram with maximal bin width smaller than the minimal distance between instances, they become ineffective.

2.1.2 Density Estimation in High-dimensional Spaces

Density Estimation in high-dimensional space poses several problems. If the dataset has a large number of attributes, it is quite likely that several of them are irrelevant and some are highly correlated with each other. A few techniques have been developed to extract such attributes from a dataset before starting the density estimation process [23][36][38]. Methods to select the relevant attributes in order to discard the irrelevant ones are called *attribute selection* or *feature selection* methods.

Moreover, high-dimensional data has some unexpected properties. With some of the attributes being irrelevant, distances between instances measured by standard distance measurements like the Euclidean distance become very similar. Every instance has almost equal distance to every other instance.

Most of a high-dimensional instance space is in fact empty. This is called the *curse of dimensionality*. It can be best explained with an example. Assume the task is to fill out an instance space, in which every dimension has been cut into ten subranges, with uniformly distributed data, so that each resulting multidimensional bin has at least ten instances. In the one-dimensional case, a dataset with one hundred equally distributed values would suffice. If the dimension of the dataset is only as high as four, already the dataset fulfilling this requirement would require a very large dataset of at least ten thousand instances.

2.1.3 Histograms

The histogram is a well-known tool for representing data distributions. Pearson is considered [56] to be the first to use the name 'histogram' in a publication in 1894. However, there is evidence that representation techniques like the histogram have been used for several centuries before Pearson named it. A histogram is built by splitting the sample range into non-overlapping intervals

also called bins. In the density histogram, the density for each bin is computed from the bin width w_i of bin_i, the number of bins k, the total number of training instances N and the training instances themselves $x_1, .., x_j, .., x_N$. With this and the indicator function I the density function $f(x)$ is defined by:

$$f(x) = \sum_{i}^{k} I(x \in bin_i) \sum_{j}^{N} I(x_j \in bin_i) \times \frac{1}{w_i \times N} \tag{2.2}$$

In this thesis only the *density histogram* is discussed. The *frequency histogram* differs from the *density histogram* in that the height of a bin of the frequency histogram is simply one unit per example that falls into the interval of the bin and does not depend on the bin width.

An important parameter of all density estimation techniques is the smoothing factor. The most common and simplest way of making histograms is the equal-width method. The range is divided into subranges or bins of equal-width. In the equal-width diagram the bin width is the smoothing factor. If the bin width is small, more irregularities can be seen and the density function is less smooth. The larger the bin width, the fewer irregularities can be seen, but some important features of the underlying distribution maybe hidden. In the method developed in this thesis, the bin widths (and with that the smoothing factor) are adapted automatically to the changes of local density and so vary over the range of values.

The smoothing factor also controls the bias-variance trade-off. If the bin width is large, the variance in the estimator is generally smaller, because the estimate is based on more data, but the bias is large. Reducing the bin width reduces the bias but increases the variance [16].

Equal-width Histograms The equal-width histogram can also be called the standard histogram technique. Equal-width histograms divide the range of the attribute into a fixed number of intervals of equal length. The user normally specifies the number of intervals as a parameter, but cross-validation can be used instead (see discussion in Section 2.1.5 below).

Equal-frequency Histograms Another basic histogram method is the equal-frequency method. Equal-frequency histograms have a fixed number of intervals and the bin width varies over the range because each bin contains the same number of data points. The number of intervals is again determined by the user.

2.1.4 Discontinuity of Histograms

A histogram represents a piece-wise constant density function. This means the function is not continuous. As long as the discontinuity of the density function is not a problem for the application at hand—this would be the case if derivatives of the density function were needed—the histogram is a potential candidate technique for the estimation of the density.

2.1.5 Evaluating Histograms

The evaluation of a histogram focuses on the quality of fit of the model to the real underlying distribution. The model is the histogram. A standard statistical evaluation criterion for probability modelling is the likelihood of the model. In this thesis the cross-validated likelihood is used for evaluating density histograms, discussed below. Another common evaluation method is the penalized likelihood and it is also briefly discussed below. However, this method is problematic in the context of histograms because histograms are not differentiable.

Likelihood

The likelihood is a commonly used measurement to evaluate density estimators [60]. It measures how likely the model is, given the data. ($X_1, ..., X_n$ are the test instances; g is the density estimator.)

$$L(g/X_1, ..., X_n) = \prod_i^n g(X_i) \tag{2.3}$$

Instead of the likelihood, the log-likelihood is often used, which gives numerical advantages because the product is transformed into a sum of terms.

$$L(g/X_1, ..., X_n) = \sum_i^n log\ g(X_i) \tag{2.4}$$

To get an unbiased estimate of the likelihood, test data that is independent of the training data has to be used. For a histogram that has n_i training instances in bin_i, let $n_{i_{test}}$ be the number of instances of the test set that fall into this bin, w_i be the bin width, and N_{train} be the total number of training instances. Then the log-likelihood L on the test data is:

$$L = \sum_i n_{i_{test}} \times log \frac{n_i}{w_i \times N_{train}} \tag{2.5}$$

Penalized Likelihood

If one of the bins' width becomes very small, the likelihood value of the histogram can grow arbitrarily large for this bin, which fact causes a problem when comparing histograms using the likelihood. The penalized likelihood tries to avoid this by putting a penalty on roughness. A density estimate is 'rough' when there are large changes in density between neighbouring areas. Silverman [60] adds a positive smoothing parameter α and with it defines the penalized likelihood as:

$$L_\alpha(g) = \sum_i^n log\ g(X_i) - \alpha R(g) \tag{2.6}$$

The roughness penalty R is a function of the density function g and is often defined using a derivative of g. For the discontinuous histogram, a different measure for the roughness value would have to be defined, or the cross-validated likelihood can be used instead.

Cross-validation

The dataset that is used to build a model is called the *training* or *construction dataset*. Testing the model on the data that it was built from is called *re-substitution testing*. If the model is fit very closely to this construction dataset, some peculiarities of this sample will be represented in the model. The re-substitution test will classify the model as good but it will not show the characteristics of the real distribution clearly. This problem is also called *overfitting*. A method that fits the data too closely can be particularly impractical if the data contains errors.

For comparison of models, it is better to use a test dataset that is independent of the dataset used for construction, but comes from the same distribution. If only one dataset is available, *hold-out* methods are applied. 'Holding out' means setting a part of the training data aside to be later used for testing.

Several hold-out methods have been developed. Sometimes only a restricted amount of data is given and no data can be spared for training. The cross-validation procedure uses all data for training and for testing in the following way. It splits the dataset into n equal-sized folds and repeats the training process n times using $n-1$ folds for training and the remainder for testing. The evaluation criterion, the so called *score function* is averaged over the repetitions.

This way the cross-validation procedure uses all instances $n-1$ times for training and once for testing. In statistics, cross-validation is commonly used to evaluate the fit of a model to the real distribution, and it is widely used in machine learning to evaluate models for classification, regression and also for clustering [61]. For the evaluation of density estimation models, the likelihood can be used as the score function.

When looking for a well-fitted model it is useful to build several models, maybe with increasing complexity, and compare them. Cross-validation can be used to decide between models and with this on the complexity of a model (i.e. as a model selection method).

The larger the value n is chosen (i.e. the more folds are used) the smaller the bias of the resulting estimate, but the larger the variance [62]. 10-fold cross-validation repeated 10 times (10×10 *cross-validation*) is commonly used to evaluate machine learning models.

Leave-one-out Cross-validation Leave-one-out cross-validation builds the model while leaving just one instance out of the training dataset. This one instance is used for testing the model. If the dataset has N instances, it is split into N folds, therefore this method is also called *N-fold cross-validation*. Leave-one-out cross-validation results in bias-less estimates [53][57]. For leave-one-out cross-validation, the training and testing is repeated N times, which means that the cost of it is high.

Cross-validated Equal-width Histograms

For histograms, the complexity can be measured by their number of bins. A variant of the equal-width histogram method selects the number of intervals—and with it a best fitting model—by using the cross-validated log-likelihood. For equal-width histograms it is not only important to select the number of intervals but also the origin of the bins [60]. The origin is found by shifting the grid by a part of the actual bin width (e.g. one tenth of it), and selecting the best one of these shifts.

For equal-width histograms leave-one-out cross-validation can be applied with no increase in cost. For each fold of the evaluation of an equal-width histogram the bins stay fixed and the log-likelihood on each test instance can easily be computed.

2.1.6 Histograms as a Visualization Tool

A common application of histograms is visualization. Nonparametric methods have been traditionally used as exploratory data analysis methods. Scott says in [56]: "A scatterplot has too much ink" and calls a histogram a scatterplot smoother. The scatterplot is a popular visualization tool. Each instance in the sample is represented as a dot. If the dataset represented is very large, entire areas can be blackened out and the information about the density of areas concealed. The histogram methods developed in this thesis can be used as visualization tools just like any other histogram method.

2.2 Tree Building Algorithms

The methods in this thesis generate histograms by constructing tree-based density functions using tree building algorithms. Tree building algorithms generate a model from data that is supported by a tree structure. These algorithms have been used for classification and regression tasks in statistics [9][26] and machine learning [46] .

To build the model, the algorithm splits the dataset recursively. Tree construction starts from the root node. Each inner node contains a split condition and corresponding to the condition the instances follow one of the outgoing branches of the node. These branches lead either to other nodes where the dataset is split again or to a leaf node. Each leaf node represents the subdataset for which all conditions from the root down to this node apply.

Most methods that can be found in the literature have simple split conditions splitting on one of the attributes only. For categorical data a split usually has as many branches as the attribute that is split on, has values. One value is assigned to each branch. Numeric data is mostly split in an axis-parallel fashion, and if the dataset is completely numeric the leaves represent rectangular subranges of the instance space. Each subrange is defined by the tests that lead from the root node to the corresponding leaf node. This set of tests can also be represented as a rule. In fact, Quinlan's system C4.5 [50] contains a function to extract rules from a given decision tree and simplifies them.

At each leaf node a simple prediction model is applied. If used for classification, this can be a single value of a categorical class attribute. For numeric classes it can be a constant or a simple function. The sum of these functions across all leaf nodes estimates a discrete or numeric function modelling the

response variable.

One of the main questions in tree construction is when to stop splitting and growing the tree [9][20][46]. The point when the algorithm cannot split a node further is when only one instance is in the node. Thus the process has to stop as soon as each leaf has only one instance. However, most of the time, splitting down to the level of individual instances results in overfitting. If the model fits the training data perfectly it is possible that it models some peculiarities of this data that do not truly reflect the underlying domain: The model is overfitted. There are other reasons for a bias towards a smaller, simpler model, like interpretability. To avoid a tree that is too complex, tree building algorithms implement more sophisticated termination criteria or use pruning techniques after training a complete tree.

Tree building algorithms stem from the much older divide-and-conquer principle, which is discussed in Section 2.2.1. In artificial intelligence, problem solving is seen as a search in the solution space. Section 2.2.2 gives a brief overview of existing search approaches in artificial intelligence. The basic principles of decision trees as far as they are relevant for this thesis are explained in Section 2.2.3. Section 2.2.4 explains how to decide between decision tree models. Section 2.2.5 discusses further details of tree pruning methods. Sections 2.2.6 and 2.2.7 discuss discontinuity and interpretability of tree structures respectively.

2.2.1 Divide-and-conquer Methods

A tree building algorithm belongs to the category of divide-and-conquer algorithms: The goal of the task at hand is tackled by reaching a subgoal first then splitting the dataset and applying the problem solving algorithm locally on the subsets and continuing these steps in a recursive fashion. This can be compared to ancient war tactics, when the troops of the enemy were split, to be overwhelmed more easily. In computer science, divide-and-conquer algorithms were designed with the hope of achieving low computational time complexity. The construction of a binary tree structure can achieve a complexity of the order $n \times log_2(n)$, which is often a much smaller complexity than the complexity that can be achieved without using a tree-based algorithm—depending on the application. The success of this tactic depends mostly on how well the task can be split into subproblems of the same size.

2.2.2 Searching in the Solution Space

The field of artificial intelligence defines *problem solving* as a search in a state space for a solution state [52]. Sometimes not all solutions are equal but a good or even an optimal solution is searched for. It is also often the case that there are too many possible solutions and the option of finding all solutions to compare them with each other is not feasible. Often the states can be pictured as the nodes in a tree and the operations as the branches in the tree: in the inner nodes subgoals have been reached and in the leaf nodes the goal. The search is done by performing operations that transfer from one state to another, starting from the root of the tree.

Search methods can be differentiated based on whether they build the tree *breadth-first* or *depth-first*. Simple breadth-first search and simple depth-first search are *uninformed* search methods. It is often more efficient to use a heuristic and perform an *informed* search with it.

As mentioned before, in most cases the solution space is too large to be searched exhaustively. Many tree-based algorithms are *greedy* because they start with an initial state, represent it as the root node and continue recursively based on locally optimal choices. Greedy methods mostly find locally optimal but not globally optimal solutions. A globally optimal solution can be found if the algorithm can fall back on earlier solutions (i.e. take back a subgoal) and explore other options from there. Taking back subgoals is called *backtracking*. However, if a greedy divide-and-conquer method without backtracking gives reasonable results, it can still be useful because of the efficiency of the algorithm.

2.2.3 Decision Trees

Probably the most prominent application of the divide-and-conquer method in the field machine learning is decision tree learning. Decision trees have been developed in parallel in statistics (Breiman et al.'s CART system [9]) and machine learning (Quinlan's ID3 and C4.5 systems [50]). Decision trees are widely used for classification tasks. In classification tasks the aim is to build a model for a given dataset that can be used for future prediction of the class labels of so far unseen and unlabelled instances. As the class value is categorical, this can also be seen as finding an approximation for a function with a discrete response variable.

Although many variants of decision trees have been developed [28] and

applied to different types of datasets, this section only discusses aspects that are relevant to the work in this thesis. For instance, missing values are excluded from consideration in this thesis.

Standard decision trees are built by splitting the instances of the training dataset into groups that exhibit purity with respect to their labels. Splits are done on one attribute at a time and each split corresponds to an inner node in the decision tree. Splits can vary mainly depending on the type of the attribute tested. In Quinlan's decision tree algorithm ID3 [50] splits on categorical attributes have one outward branch from the node for each attribute value. ID3's splits on numeric attributes are simple two-way splits based on inequalities comparing to a constant, where the split value is stored at the node. Instances with a value smaller than the constant concerned follow down the left branch of the node and all others instances follow down the right branch. The leaves contain all instances to which the corresponding conjunction of tests applies.

Tree building algorithms automatically ignore attributes that are not relevant to the classification. This makes this method very suitable for high-dimensional datasets. In fact, a decision tree can be used for attribute selection before performing classification.

Split Selection

An important question in decision tree construction is how to select splits for the inner nodes of the tree in order to get a concise and predictive tree. The aim is to reach pure nodes as quickly as possible. Quinlan's decision tree algorithm ID3 [50] uses *information gain* to decide which attribute to take for the next split. The information gain measure is based on the *entropy* value. Entropy has been defined in information theory and is a measure for the expected encoding length if the information is represented in bits. A bit is an item that can be in a binary set of states, for instance can have 0 or 1 as its values. The entropy of a collection of instances C depends on the proportion of instances of each class p_i. The total number of classes is c. The entropy is defined as:

$$Entropy(C) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{2.7}$$

The information gain is a difference of entropies. For the decision tree building process it is of interest how much the entropy decreases after per-

forming a split on the attribute A. The information gain shows how much purer the subsets get after the split compared to the dataset before the split. Therefore the information gain (IG) at an inner node depends on the collection of instances C and the attribute A that the split is performed on. It is given by:

$$IG(C, A) = Entropy(C) - Entropy_{afterSplit}(C, A) \tag{2.8}$$

The entropy after the split is the sum of the entropies of the subsets, with each of these subset entropies weighted by the number of instances in the subset. Let v be the number of values of the attribute (or two if the split is numeric), and C_i be one of the subsets of instances after the split. Then the expected entropy after splitting is given by:

$$Entropy_{afterSplit}(C, A) = \sum_{i=1}^{v} -\frac{|C_i|}{|C|} Entropy(C_i) \tag{2.9}$$

The next question is when to stop expanding the tree. This rule is often called the *stopping rule*. A very simple stopping rule is to enforce a certain minimum number of instances at each leaf node.

In general, the search through all possible split combinations is computational infeasible and this is why greedy methods are applied for the search. Since a greedy method is used, the resulting tree is often not optimal. Therefore, various methods have been developed to improve the tree after it has been built. This can involve pruning the tree or performing a more drastic restructuring. Tree pruning is important to avoid overfitting. It is explained in more detail in Section 2.2.5. The next section first explains a way to evaluate and compare tree models, which can be used for pruning.

2.2.4 Evaluating Decision Trees

For classification algorithms, one of the criteria used to decide between models is classification *accuracy* [46][53]. Accuracy is a measure that tells the scientist how well the classifier's model can predict class labels of instances correctly [24]. A prediction is counted as correct when the predicted class is the same as the actual class value in the test dataset that is used to estimate accuracy. Let n be the number of instances in the test dataset, $X_1, ..., X_n$ be the attribute vectors of the test instances, $c_1, ..., c_n$ be their actual class values, $\hat{g}$ be the classification model, and I be the indicator function. Then accuracy

is defined as follows:

$$Accuracy = \sum_{i=1}^{n} I(c_i, \hat{g}(X_i)) \tag{2.10}$$

Sometimes, instead of accuracy the *error rate* is used, which is the opposite of accuracy:

$$ErrorRate = 1 - Accuracy$$

Cross-validation of Decision Trees

The accuracy measured using the training data (*re-substitution accuracy*) does not give a reliable estimate of accuracy on future data (see also the discussion in Section 2.1.5). In statistics and machine learning, cross-validation is widely used to evaluate models. 10-fold cross-validation (repeated ten times) is commonly used to evaluate machine learning models. For classification methods, accuracy is a popular score function.

2.2.5 Controlling the Size of the Tree

Large trees are often inferior models because they are more likely to overfit the training data. Overfitting means that the model is under-smoothed. It increases variance in the estimate and can reduce predictive performance.

To avoid overfitting, constructing complex large structures should be avoided. The bias towards simpler models follows the principle of *Occam's razor* (see the subsection below). Large trees can be avoided when tree construction is stopped at a certain tree size. The tree can also be fully expanded first and then *pruned* back in size. Pruning generally means to cut off branches of a tree. Sometimes the tree is also restructured more profoundly. This section only discusses the basic methods of pruning, which are represented by the two main approaches *post-pruning* and *pre-pruning*.

Post-pruning *Reduced error pruning* [20] is an example of a post pruning method. The decision tree is first built to its full size. During training, a subset of the training dataset is set aside as a *validation set*. At pruning time, each node is then tested to see whether removing it would increase accuracy on the validation set. Nodes are removed from the tree until accuracy stops improving.

Pre-pruning Strictly speaking, pre-pruning [20] does not perform pruning. It means to stop building the tree instead of removing nodes after building the tree. A simple pre-pruning technique uses a threshold for the information gain. If the information gain from splitting at a node is below this threshold, then the node stays as a leaf node and is not split further.

Occam's Razor

Explaining it in a simplified way, Occam's Razor stands for preferring simple models over more complicated ones and is often applied as a rule of the thumb. It got its name from the logician William of Occam, who lived in the 14^{th} century. Occam's razor was not invented by him as such. The term was used later in the 19^{th} century for methods that favour simpler theories over more complex ones. In the case of decision trees, applying Occam's razor means to prefer smaller trees over larger ones, as long as they are a sufficiently accurate model for the distribution of the data. Thus, pruning is an implementation of Occam's Razor.

2.2.6 Discontinuity of Tree Structures

Tree structures or the functions represented by them are generally not continuous but mostly piecewise constant. Although this is not discussed in this thesis, there are ways to make them smooth. These methods could be adapted for the methods presented in this thesis but this is left as future work. Smoothing methods generally reduce interpretability.

2.2.7 Interpretability of Tree Structures

For a one, two and even a three-dimensional dataset, the resulting partitioning of the tree construction process can easily be presented in a graphical way. If the dataset is of higher dimensionality, this is not as easily possible.

Fortunately, every decision tree is essentially a set of rules. Rules are considered to be a human-interpretable model. One rule corresponds to one leaf, where the tests that are in the nodes on the way from the root node down to the leaf node form the conditions of the rule.

Moreover, the splitting values of numeric variables that occur in the tree provide definite numeric values that help with interpretation and these values could be taken as splits between 'low' and 'high' values (or between 'low', 'medium' and 'high' values, etc.).

2.3 Related Work

First discussed below is Fayyad & Irani's discretization algorithm [18] for continuous attributes. The method of splitting a range into subranges, as a preprocessing method for classification tasks, is generally called discretization. Two subsequent sections discuss two density-based clustering algorithms.

2.3.1 Fayyad & Irani's Tree-based Supervised Discretization Method

Fayyad & Irani invented a discretization method [18], that was developed to deal better with numeric attributes in decision trees. It can also be used to prepare the attributes before tree building starts. Discretization transforms a continuous attribute into an ordered categorical one by splitting the range of the values into subranges. When building a decision tree, a continuous attribute is split by identifying pure subsets of data with respect to the class attribute.

Like ID3 [50], Fayyad & Irani's method applies the splits recursively, starting from the complete datasets. For each split the entropy is measured between each of the possible partitions and for each of the attributes. The split with the best *information gain* is selected in a greedy fashion for the next split. Information gain is computed as the difference of the class entropy of the dataset before and after the split.

A criterion based on the minimum description length principle [39] [51] [64] is used to decide when to stop splitting (i.e. for pre-pruning). It defines the problem of deciding, whether a cut improves a partition or not, as a coding problem. The cost of sending the uncut partition is compared with the cost of sending the split partition. The cost of sending the uncut partition consists of the cost of sending all N class labels, which, depending on the entropy in the set, have an average code length of l. Additionally a code book has to be

transmitted with one code for each of the k different classes. Cost C_{UNCUT} is therefore:

$$C_{UNCUT} = (l \times N) + (l \times k)$$

After the cut the dataset is split into two subsets in which the number of examples is N_L and N_R. Further information about the split is the cut value of the split which can be encoded with $log_2(N-1)$ bits. With this, C_{CUT}, the cost of transmitting the split data is:

$$C_{CUT} = (l \times N_L) + (l \times N_R) + log_2(N-1)$$

This way of selecting the number of splits can be classified as pre-pruning method which is based solely on the attribute concerned and the class attribute. In a more recent work, Jin and Breitbart [32] defined a more generalized entropy for discretization and also implemented it with a new dynamic programming algorithm.

2.3.2 The Density-based Clustering Method STING

Clustering is a form of unsupervised learning, where the aim is to find clusters in the data. Density-based clustering methods define clusters as areas of high density and apply density estimation to find these in the distribution of the data.

The density-based clustering method STING [66] (STING stands for STatistical Information Grid) partitions the instance range into equal-sized multidimensional boxes similar to an equal-width histogram. In the clustering literature, this is also called a grid-based method. The grid is on several levels and the upper-level boxes are cut into lower-level ones. This way one upper-level box is always connected to its lower boxes like the nodes in a decision tree but with the areas cut in all dimensions. Each box stores statistical information like the mean and the standard deviations of the values falling into the box. The corresponding statistical values of the upper-level boxes are efficiently computed from the lower-level boxes they contain. This computation is independent between boxes at the same level and so can be done in parallel. STING was designed primarily for accelerating data base access for query answering.

2.3.3 The Density Based Clustering Method CLIQUE

The density based clustering method CLIQUE [2] is, like STING, also a grid-based clustering method, but specifically designed for high-dimensional data. Again, the instance range is split into equal-size boxes similar to an equal-width histogram, but CLIQUE starts from a one-dimensional partition. Highly-dense areas in this one-dimensional partition are marked for exploration in further dimensions. Dimensions are only added if they result in a division into high-density areas and low-density areas. In this way, irrelevant attributes are filtered out. Methods like CLIQUE are also called *subspace clustering* methods.

2.4 Summary

The work in this thesis is on building density estimation trees. The basic concepts for this work are density estimation and tree building algorithms. Density estimation has a long history in statistics. Tree building algorithms were developed, in parallel, in statistics and machine learning. The algorithms developed in this thesis result in a one- or multidimensional histogram and can be classified as nonparametric density estimation methods. Nonparametric density estimation methods model complex distributions well but, for most existing methods, density estimation in high-dimensional spaces causes problems.

Tree building algorithms are popular for their positive computational behaviour. A binary tree structure built in a greedy fashion can reduce the computational cost of building and of the usage of the model (e.g. prediction for classification). The tree building process selects only relevant attributes and is therefore well suited for high-dimensional datasets.

Histograms and tree models in general have the property of good interpretability. A drawback is sometimes the discontinuity of their resulting estimated function.

Cross-validation, which will be used extensively in parts of this thesis, is a commonly-used technique to evaluate the predictive accuracy of classification algorithms in machine learning [46][53] and can also be used to evaluate the fit of a statistical model. Moreover, it can be employed to choose between models and to select the complexity of a model.

The next chapter explains the technical details of the TUBE algorithm for the univariate case. In the machine learning field, the splitting of a range

into subranges is often referred to as discretization. Univariate TUBE will, as a first application, be evaluated as a discretization technique and secondly will be applied to a known classification task as a discretization technique for density estimation.

Chapter 3

Univariate Density Estimation Trees

The methods developed in this thesis build a tree structure as a model for the density distribution of the data. This chapter discusses the density estimation method proposed for the simple univariate case. For this purpose data was used with only one numeric attribute. The algorithm that is presented is called TUBE (Tree-based Unsupervised Bin Estimator) [54]. The name TUBE is further explained in Section 3.1. The algorithm was also adapted to the multidimensional case, this algorithm is discussed in Chapter 5.

TUBE's model for the density distribution is a histogram. Histograms as a technique are about 400 years old [56]. A classical histogram is a very simple method for density estimation. With a given set of sample values the range of the sample is cut into non-overlapping subranges, also called bins, of equal length. Over each subrange, a rectangle is drawn whose height specifies the number of instances falling into the bin. For each instance, one unit of height is added. This type of histogram is still used as a *frequency histogram*. The method presented here produces subranges of various lengths, therefore the *density histogram* method is applied. For the *density histogram* method, the height of each bin represents the density associated with the bin. The height h_i of each bin_i is computed using the bin width w_i, the number of instances n_i falling into bin_i, and the total number of instances N in the dataset. With this, the height h_i of bin_i is (Equation 2.2 from Section 2.1.3 repeated; note height h_i is equal density d_i):

$$h_i = \frac{n_i}{w_i \times N}$$

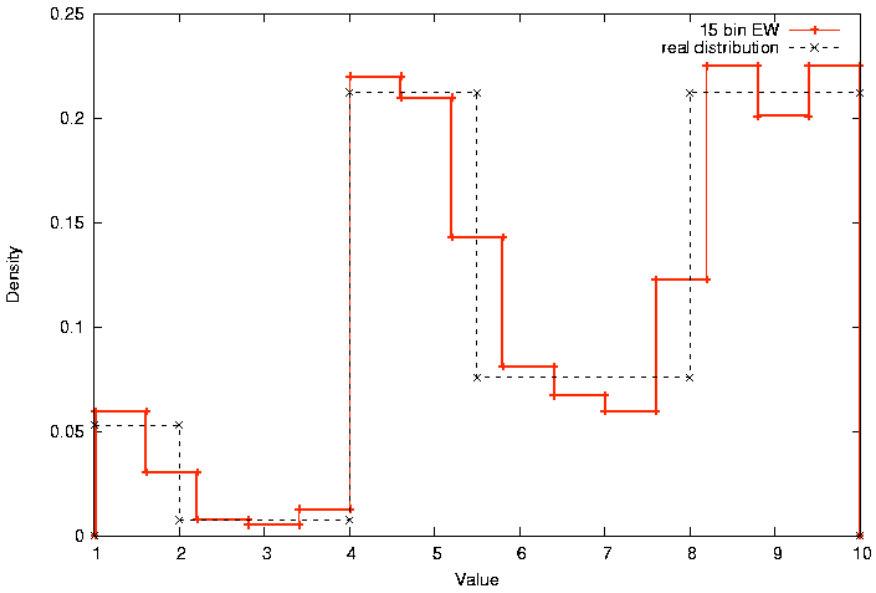

Figure 3.1: Example dataset split with equal-width binning (Fifteen bins).

Having intervals of varying length makes it possible for the TUBE algorithm to adapt the bin width to the change in local density. TUBE has the goal of cutting the range and splitting the dataset in such a way that intervals are defined which exhibit uniform density. Of course, in practical problems the true underlying density will not really be uniform in any subrange, but the most significant changes in density should be picked up and should result in separate intervals.

The algorithm uses a tree-based algorithm to determine the cut points. More specifically, it builds a density estimation tree in a top-down fashion. Each node defines one cut point between two subranges or bins. At the leaves, Equation 2.2 for the density estimation function is applied. The sum of leaves defines the density estimate and can be drawn as a density histogram, which graphically represents the density function.

3.1 Tree-based Unsupervised Bin Estimation

In contrast to the classical equal-width histogram, TUBE divides the range of an attribute into intervals of varying length that are adapted to the local

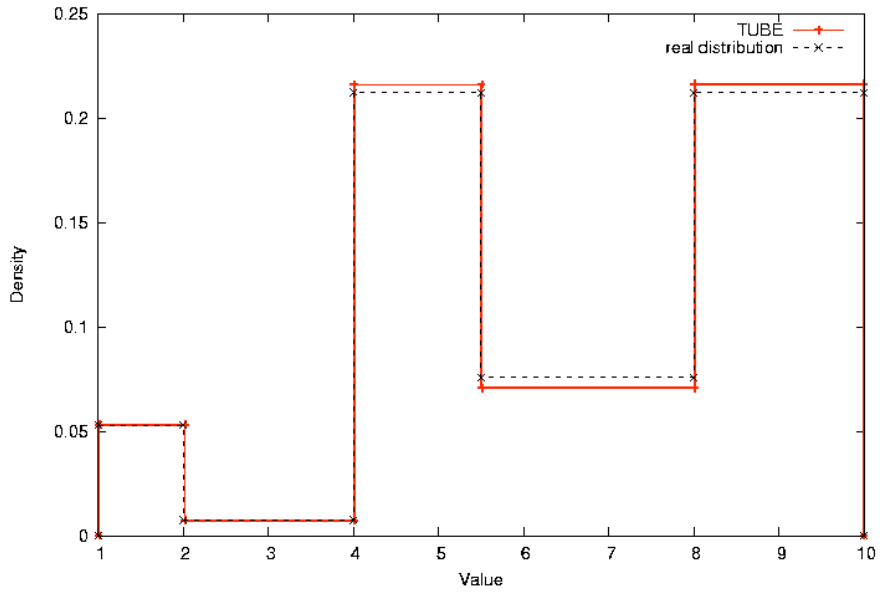

Figure 3.2: Example dataset split with TUBE.

density. Figure 3.1 shows an equal-width estimator (with fifteen bins) for a simple artificial dataset and Figure 3.2 the TUBE binning for the same dataset. In both figures the 'true' density (the density function that was used to generate the data) is plotted with a dotted line. For the model building process, the sample data can be called training data—as it is for classification models in machine learning.

TUBE uses a top-down tree-based algorithm to find the cut points which cut the range into bins. This means it first cuts the full range into two subranges and then repeats this process recursively. Every value in the sample data represents a point on the real line. Between every two adjacent points a cut (at a cut point) is attempted. The best of these cut points is taken to perform the division into subranges. How the best pair of points (i.e. instances) are found is explained in Section 3.3. Where exactly to cut the range in the empty space between these two points is discussed in Section 3.4. The full recursive algorithm is discussed in Section 3.5. Section 3.6 explains how TUBE controls the size of the tree.

Every discretization method needs to sort the values before starting with the binning. TUBE sorts the attributes using a quicksort function.

TUBE is an *unsupervised* method. *Supervised* methods use class labels for their computations. Fayyad & Irani's supervised discretization method [18], discussed in Section 2.3.1, splits the range of a numeric attribute into subranges in preparation for classification tree building. The method is supervised because it takes the values of the categorical class attribute into account. TUBE is also a tree-based discretization method similar to Fayyad & Irani's discretization method, but is *unsupervised* and does not use the class of the instances—only the values of the attribute—to determine the subranges.

3.2 Evaluation of a Binning

When building a density estimation tree, the algorithm has to repeatedly decide between different alternatives. It has to decide where to split the dataset, at which value to cut exactly, and when to stop cutting. To perform these decisions, an evaluation technique is needed.

In this thesis the likelihood and the cross-validated log-likelihood criterion are used to evaluate the TUBE binning and to estimate parameters for the density model. This principle for estimation is called *maximum likelihood estimation* [26]. The aim is to maximise the likelihood of the parameters for the estimator considering the sample dataset. As mentioned in Section 2.1.5, the likelihood L depends on the test values (instances) $X_1, ..., X_n$ and the density function $g(x)$. For numerical reasons the log-likelihood is used instead of the likelihood. As stated before, the log-likelihood $LL(X_1, ..., X_n)$ is defined as:

$$LL(X_1, ..X_n) = \sum_{i=1}^{n} log\ g(X_i)$$

For histograms, $g(x)$ is the height of the bin that contains the value x (see also Equation 2.2). Therefore, for a density histogram with k bins, with $n_{i_{test}}$ being the number of test instances falling into bin_i, n_i being the number of training instances in bin_i, and w_i being the width of bin_i, the log-likelihood LL of the test instances using the histogram as density estimator is:

$$LL = \sum_{i=1}^{k} n_{i_{test}} \times log \frac{n_i}{w_i \times N} \tag{3.1}$$

The log-likelihood is sometimes also referred to as cross-entropy [26].

3.2.1 Adapting for Empty Bins

The log-likelihood evaluation function in Equation 3.1 cannot be evaluated in cases when a test instance falls into a bin that remains empty during training: $n_i = 0$ means that $\frac{n_i}{w_i \times N}$ equals zero and the logarithmic function for the value is undefined. To solve this problem, a single hypothetical instance is spread over the whole range of the data by adding a part of the instance to each bin that is equivalent to the relative width of the bin. Let W be the total length of the range, then the above equation becomes:

$$L = \sum_i n_{i-test} \times log \frac{n_i + \frac{w_i}{W}}{w_i \times (N+1)} \tag{3.2}$$

3.3 Finding the Split Point

The algorithm starts by splitting the full range of values once and then recursively continues by splitting the subranges. As preparation for the search for the best split point, the values are sorted. With the first split, the range is divided into a simple two-bin histogram and the quality of the split measured with the likelihood. This process is repeated recursively.

The main question is how to choose the split point. This decision is based on the likelihood criterion, evaluated on the training data. With n_{left} and n_{right} being the number of instances in the left and right halves, and w_{left} and w_{right} being the widths of the two halves, the log-likelihood LL of a split is then:

$$LL = n_{left} \times log \frac{n_{left}}{w_{left} \times N} + n_{right} \times log \frac{n_{right}}{w_{right} \times N} \tag{3.3}$$

The question that still remains is where exactly between two points should the cut be set. This is discussed in detail in the next section (Section 3.4). It explains why TUBE actually considers two cuts in each empty space between each of two neighbouring points, and also a cut to the 'left' of the smallest value and a cut to the 'right' of the largest value if there is empty space at the ends of the range. This means that TUBE actually computes the log-likelihood of $2 \times (N-1) + 2$ possible splits before finding the best split point for N instances.

3.3.1 Examples of Split Point Selection

This section gives some examples of how TUBE finds the first split point in a range of values. For these examples, four univariate datasets have been generated. On each of them, a split into two bins is performed. Each diagram shows the distribution of the data as it was generated as a dotted line (the 'real' distribution), the resulting two-bin histogram (after the first split), and a plot of all log-likelihood values computed. The irregularity in the log-likelihood curves stems from the locally irregular distributions of the data and from the fact that each empty space between two successive points was attempted to be cut twice, once setting the cut point close to the left point and once close to the right point. The log-likelihood value for each of these splitting attempts was evaluated and plotted.

Split 'Step' in Distribution The first example dataset has a distribution that consists of two ranges with different uniform densities. Figure 3.3 shows the step-like distribution function with which the data was generated as a dotted line. The log-likelihood data is plotted along the range of the values and shows a peak where the step is in the distribution. The split is selected where the log-likelihood is maximal and the resulting two-bin histogram is also shown in the diagram. Note that the 'real' distribution and the resulting 2-bin histogram generated with TUBE are very similar.

Split on Several 'Steps' The example dataset in the diagram in Figure 3.4 has a 'step'-wise uniform distribution with five points where the density changes abruptly. The steps are of equal height. The log-likelihood curve and the resulting two-bin histogram show that the TUBE algorithm selected the middle step for splitting.

Split on Gaussian Data Figure 3.5 shows data that corresponds to a Gaussian distribution. The log-likelihood curve shows a maximum at the left lower bend of the curve and the cut is performed there.

Split to Cut Out Empty Range The dataset for this example (Figure 3.6) has an empty subrange between two equally and uniformly distributed areas. The log-likelihood curve has two peaks at the 'edges' of the empty space. One of them is slightly higher since all data values are randomly generated

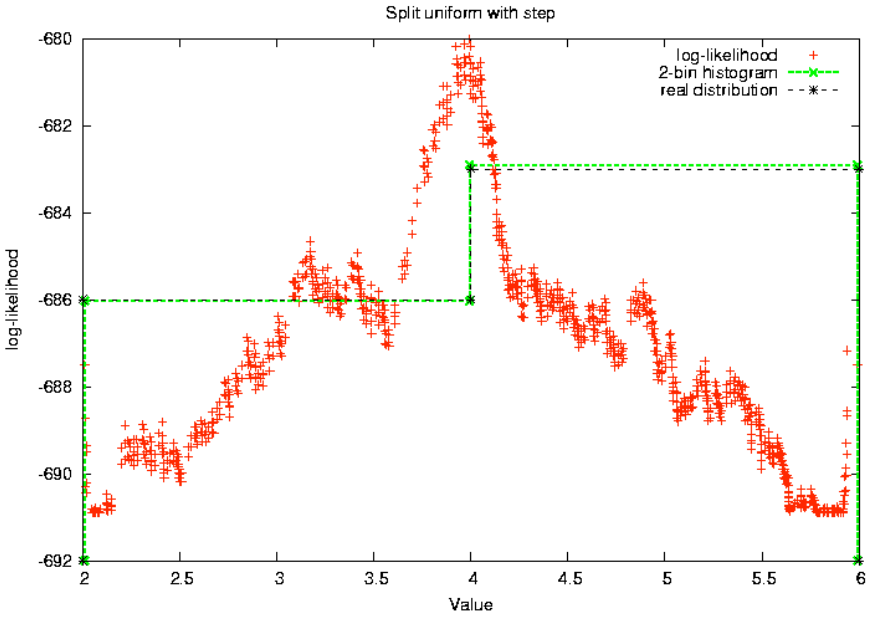

Figure 3.3: Splitting a one-step uniform distribution.

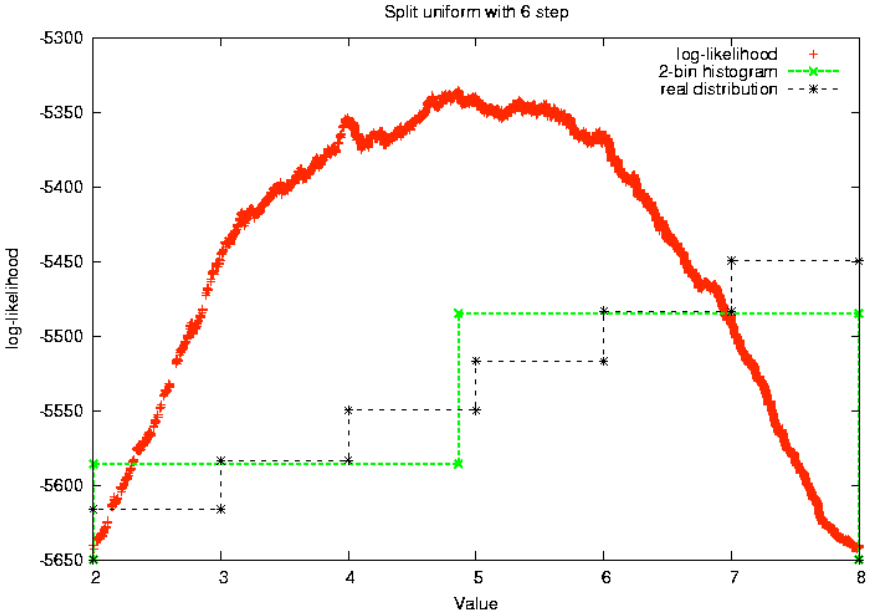

Figure 3.4: Splitting a multi-step uniform distribution.

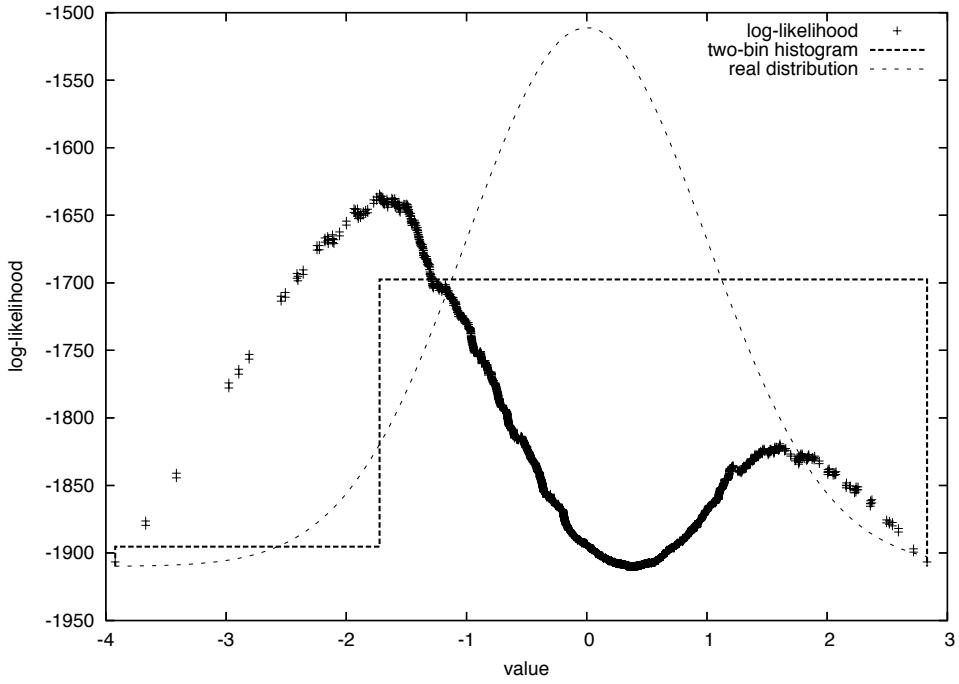

Figure 3.5: Splitting a Gaussian distribution.

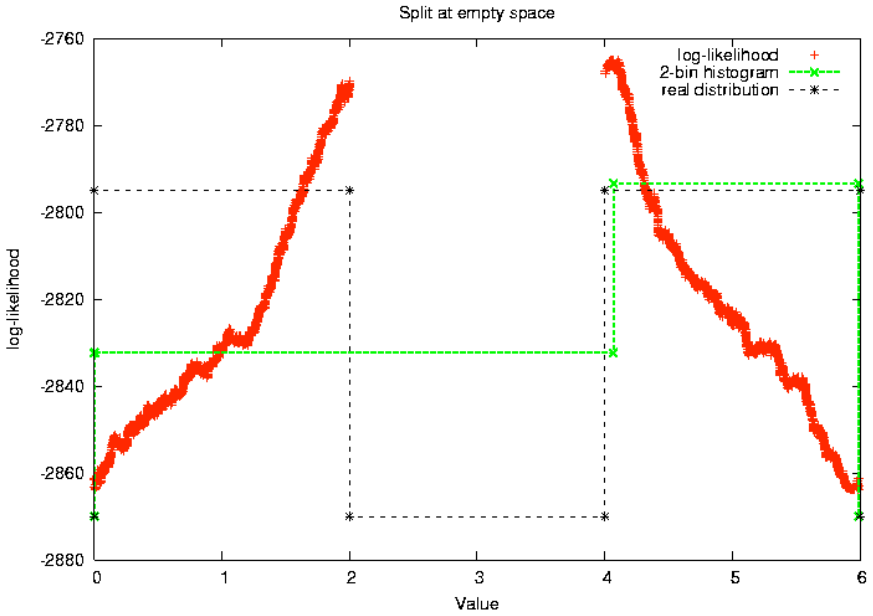

Figure 3.6: Splitting at an empty subrange.

and therefore the two sides are not completely identical. The histogram in Figure 3.6 shows that the right edge is chosen for splitting.

3.4 Where to Set the Cut Point for the Split

The previous section (Section 3.3) explains the basic method for finding a split point. However, the question is not only between which two successive points to split but also where in the empty space between the two points the cut should best be made. When comparing TUBE's method of splitting with the splitting of a numeric attribute for decision tree building (see discussion in previous chapter, Section 2.3.1) it can be seen that in Fayyad & Irani's method [18] the value of the splitting criterion does not vary over the empty range between points. This is because in decision tree learning the splitting of a numeric attribute depends only on the class values of the instances to the left and to the right of the cut. These values do not depend on where exactly in the empty space between the two points the cut is performed. In contrast, TUBE's unsupervised split criterion depends on the total number of instances to the left and to the right of the split and also on the width of the resulting subranges.

It can be shown that the likelihood is always lower when splitting midway between two consecutive instances than when splitting closer to these or at the points. If the range to be split consists of two uniformly distributed subranges s_1 and s_2 with densities $d_1 < d_2$ and the instances of subrange s_1, ordered by their values, are $x_1, x_2, ..., x_i$ and the instances of subrange s_2 are $x_j, ..., x_{N-1}, x_N$ then the difference between the densities—and therefore also the log-likelihood of the cut— is always larger with a cut closer to x_j because it makes the density d_2 even larger. If the densities d_1 and d_2 are in fact the same, then a cut closer to either x_i or x_j makes the densities different which as well increases the likelihood of the cut.

3.4.1 Example: Cutting Around an Empty Space

The dataset for this example was generated with three subranges: The areas to the left and right of the empty space have equal width and an equal number of instances and therefore exhibit equal density. Figure 3.7 plots the log-likelihood at the two instances that correspond to the left and right end points of the empty space (circles) and the log-likelihood at nine points between the

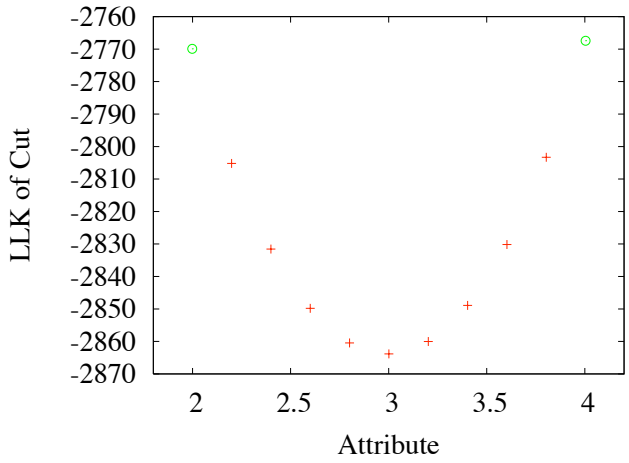

Figure 3.7: The log-likelihood is minimized between instances.

values (crosses). As can be seen, the log-likelihood varies over the range. It has a maximum at the end points of the empty space and a minimum in the middle of the empty range.

3.4.2 Placing the Actual Cut

Based on the above example it can be seen that every training instance defines two potential maximum likelihood cut points because the log-likelihood of a division into two bins is maximised at the end points of the empty range and actually has a local minimum between the two adjacent points. This means that the cut be at one of the instances—and not between instances as in the case of decision trees for classification—and set the bounds of the resulting bins to include the corresponding instance in either (a) the left or (b) the right subset.

However, cutting at the actual values can lead to problems in datasets where several instances have the same value. It is not possible to have a bin of width zero because the bin would have infinite (actually undefined) height. Therefore, the actual cut is performed close to the point concerned and not exactly at the point. To implement this, a fixed cut distance to the points is set and the algorithm considers cutting twice in each empty range, namely before and after each value, and also at the beginning and end of the full range—if the empty space at the end is larger than the fixed cut distance. However, if the distance between two instances is below the cut distance, the cut is set in the middle between these values.

In the implementation of TUBE, which is used for the experiments pre-

sented later, the cut distance has the default value 10^{-5}, but can be set as a user parameter. The default value was used in most experiments that are presented in subsequent chapters.

3.5 Building the Density Estimation Tree

This section explains how TUBE finds a set of k split points which form a good histogram. Such a histogram is a valuable density estimator. TUBE's tree building method first divides the whole range into two bins and continues recursively on the subranges of the newly formed bins. The previous sections discuss in detail the subtask of finding the best cut point in a subrange. Two more questions are left to be answered: In which order are the subranges divided and how does TUBE control the size of the tree? Or, to summarise these two questions, how is the tree structure built?

The selection of k cut points can be seen as a problem for which a solution is found by searching through a solution space. A partially or fully built tree is a state in the solution space. Each binning with k cuts is a possible solution. In how many ways can a range be divided recursively into $k+1$ bins with k cuts, when it has n possible cut values? To cut a range into $k+1$ bins TUBE builds a tree with k inner nodes. For this, it selects k different values of the n possible cut points in the range. For each selection of k cut points TUBE again can build several different trees depending on the order these k cut points are applied. How many different trees can TUBE build from k cut points? This number is expressed with the *Catalan numbers* [35]. The k^{th} number of the Catalan series is defined as:

$$C_k = \frac{(2k)!}{(k+1)!k!} \tag{3.4}$$

The Catalan numbers do not count trees with identical structure, which is what applies to TUBE's trees as TUBE always has the smaller values on the left of an outgoing branch, building ordered trees. The Catalan series shows that for a binning with ten cuts the number of possible trees is 16796. The numbers increase very quickly with increasing k. The 20^{th} number in the series is already larger than 6×10^9.

Hence, even ignoring selection of the cut points, it is clear that in most cases the set of all possible trees is too large to be built and evaluated. A search method must be applied to find a good solution with reasonable effort.

Expanding a node by performing a cut is an *operation* to change from one state to another in search of a good solution.

Tree-based search algorithms vary in the order in which they expand nodes and grow the tree. Standard ordering strategies are the breadth-first strategy, which means expanding all nodes of one level before expanding further children, and the depth-first strategy, which means expanding children recursively (usually from left-to-right) before expansion of further nodes on the same level. Both these methods are *uninformed* ways of expansion [24]. In contrast, TUBE implements best-first expansion of nodes. A best-first search is an *informed* search method [24]. The log-likelihood values of the splits represent the information used in the informed search and help the algorithm to find a good solution more efficiently than an uninformed search.

TUBE selects the next best node to be expanded using the log-likelihood criterion. As soon as a split is performed, the new subranges are examined for their best local split. These locally optimal splits are not immediately performed but evaluated according to their log-likelihood improvement and stored in an ordered *priority queue*. When a new split is to be performed (i.e. a node is to be expanded) the first element is popped off the queue and that split is performed. The pseudo code of the TUBE algorithm is shown in Algorithm 1.

The selection of the next node to expand is a heuristic approach. The overall algorithm is a greedy search that does not find the globally optimal k-way division. But it is preferable over the full search of the state space for a solution, because it finds a division that is a computationally inexpensive estimate when a full search is computationally infeasible. As mentioned above, greedy search methods are well know in supervised learning e.g. for building decision tree models. TUBE applies greedy search to unsupervised discretization. TUBE's tree building algorithm is a greedy method that takes the best solution found at the current point in the construction process and does not look back to see whether a better one could be found after the next step. This means it does not perform *backtracking*.

With the steps explained above, the TUBE algorithm does not stop growing the tree until all subranges contain a single value (i.e. it overfits). To avoid overfitting, tree growth must be controlled using a stopping criterion. The simplest stopping criterion is based on setting the maximal number of cut points per user parameter. This would be a global stopping criterion. TUBE implements a further global and also a local stopping criterion. Both are

Algorithm 1 Pseudo code for the basic binning algorithm.

```
maxNumBins ⇐ CV-ForOptimalNumberOfBins();

numSplits ⇐ 0;  {counts current number of splits}
splitPriorityQueue ⇐ empty;  {priority queue stores next possible splits}

firstBin ⇐ new Bin;  {bin which contains the whole attribute range}
binList.add(firstBin);  {list to gather all bins}
split ⇐ bin.findBestSplit();  {find the best split in the range of the bin}
splitPriorityQueue.add(split);

while numSplits + 1 < maxNumBins do
  nextBestSplit ⇐ splitPriorityQueue.top();  {best split in queue}

    {perform the split on the bin}
    {and replace the bin in the bin list with two new bins}
  newBinLeft, newBinRight ⇐ nextBestSplit.performSplit(binList);
  numSplits + +;  {one more split done}

    {finds the best possible split in the range of the new left bin ..}
  split ⇐ newBinLeft.findBestSplit();
  splitPriorityQueue.add(split);  {adds it to the priority queue}
     {.. and in the range of the new right bin}
  split ⇐ newBinRight.findBestSplit();
  splitPriorityQueue.add(split);  {adds it to the priority queue}
end while
return binList
```

explained in the next section.

3.6 Stopping Criteria

Stopping criteria for tree building algorithms differ in the way they are applied, locally or globally. A local criterion is applied at every node and stops splitting a node as soon as the criterion is met. If a global criterion is applied, it is evaluated on the whole binning after every cut. In TUBE, both types of methods are implemented and the user can choose per user parameter between them. TUBE's local criterion evaluates a penalty value. If the improvement in log-likelihood for the current node is not over a certain threshold, the splitting is not performed. As a global criterion, TUBE uses cross-validation to determine the number of bins for which the cross-validated log-likelihood does not increase further.

The method using the local criterion is a *prepruning* approach. It decides to stop expanding instead of pruning back after expanding. The global method investigates binnings with up to $N-1$ split points (when N is the number of instances in the dataset) before deciding on the best number of bins. It is a *postpruning* method.

3.6.1 Local Stopping Criterion: Penalty Rule

By proceeding solely based on maximising the likelihood of the training data, the algorithm would not stop cutting until all subranges contain a single value because the likelihood of the binning improves for each cut, and only stays the same in the rare and not realistic case of totally uniformly distributed data. To find a non-overfitting density estimate Fisher [60] introduces the *maximum penalized likelihood* (see also Section 2.1.5), which subtracts a penalty from the log-likelihood of an estimated density function. The penalty value is relative in its magnitude to the roughness of the density estimate. To measure the roughness, the derivatives of the density function are often used. As histograms represent density functions, that are piecewise constant, a derivative cannot be formed.

Fayyad & Irani [18] have developed a local stopping criterion which is based on the Minimum Description Length (MDL) principle. TUBE implements a stopping criterion following this example. A split is only accepted as the best one in a subrange if the improvement in log-likelihood is larger than a threshold. The MDL threshold is based on coding theory and given by the minimal possible encoding of the classifier. TUBE's MDL penalty is given by the minimal encoding of the split based on the local number of instances n and the value 2 for encoding the cut value. Fayyad & Irani also add the 'codebook', which in the supervised case depends on the number of classes. However, since the method considered here is unsupervised, the number of classes is irrelevant. With this penalty the threshold P at a node is:

$$P = -log(n) - log(2) \tag{3.5}$$

The penalty is applied each time a minimum in a subrange is found. If the improvement in log-likelihood resulting from the split being considered is below the threshold, the split is not performed and the splitting stops.

3.6.2 Global Stopping Criterion: Cross-validation

A global stopping criterion is a measure computed from properties of the whole binning. TUBE uses the 10-fold cross-validated likelihood as the global stopping criterion. The log-likelihood is evaluated based on each of the ten test sets in the 10-fold cross-validation. The tree building algorithm is applied to the corresponding training data. This is first done with the uncut dataset so the algorithm can also recognize if no cut at all should be performed. Then the maximal number of cut points is increased in increments of one. This can be implemented efficiently: to find k cut points, one can use the division into $k-1$ cut points and add one more. By default, the algorithm iterates up to $N-1$ as the maximal number of cut points (i.e. the cross-validated log-likelihood is computed for all trees with 1 up to $N-1$ cut points, when N is the maximum number of training instances in the ten training sets). For each of the $N-1$ iterations, the algorithm computes the average log-likelihood over the test folds and from this the number of splits that exhibits the maximum average value is chosen.

Note that this method involves growing a density estimation tree eleven times: once for each of the ten training folds, and finally for the full dataset based on the chosen number of cut points. Nevertheless, the time complexity of the binning algorithm remains $O(NlogN)$ because cross-validation introduces a constant factor only.

As mentioned above, the algorithm decides which node to split next using best-first node expansion. If the stopping criterion is global, as it is in the cross-validation-based criterion, the order of node expansion is important and the best nodes need to be expanded first. That is why best-first node expansion is used.

3.7 Example: Tree Generation using TUBE

This section presents an example of the construction process using the dataset shown in Figure 3.8, where the real distribution of the generated data is plotted as a dotted line and the histogram constructed by TUBE as a full line. For this example, the simple way of cutting at the actual end points of an empty range, instead of twice in a fixed distance from these points, is used (see discussion in Section 3.4).

First, the best cut point is found in the whole range and two new bins are

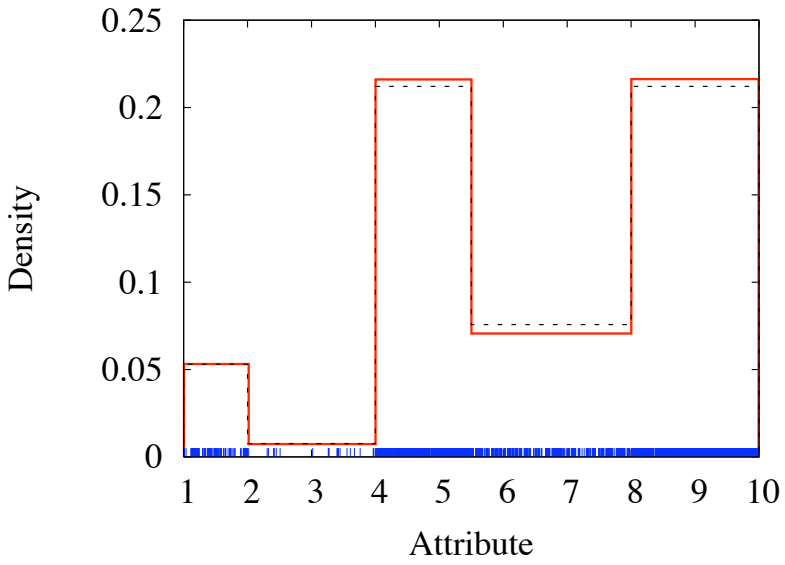

Figure 3.8: TUBE chooses five bins of varying length.

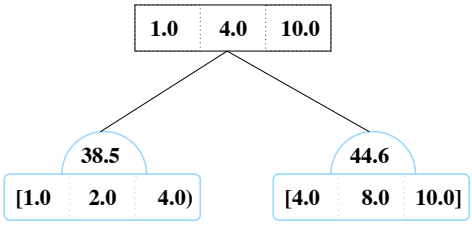

Figure 3.9: Tree after the first cut.

formed. Within the two subranges, two new locally optimal cut points are searched for. Both splits are evaluated and the improvement in log-likelihood for the division into the resulting three bins is computed for both possible splits. Figure 3.9 shows the tree corresponding to this situation. The root node represents the first cut and the two leaf nodes represent the next two possible cuts. (All values are rounded.)

The root node represents the whole range, and all nodes further down the tree a subrange. The values given in the left and right part of the rectangle corresponding to a node represent the minimum and maximum of the subrange. The overall minimum and maximum of this example dataset are 1.0 and 10.0 respectively. Each leaf node represents a bin and the given range exhibits a "[" if the minimum value itself is part of the bin and a "(" if it is part of the next bin. The notation for the maximum is analogous. The variable written in the middle of the node represents the cut point.

The whole range is first cut at the value 4.0. The next possible cut points

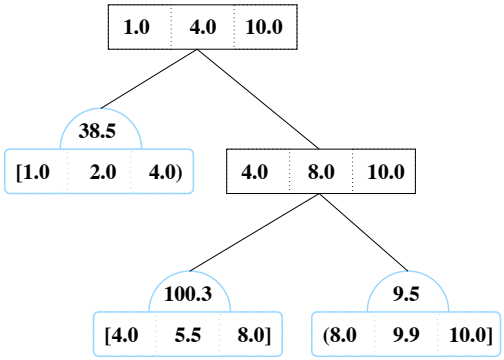

Figure 3.10: Tree after the second cut.

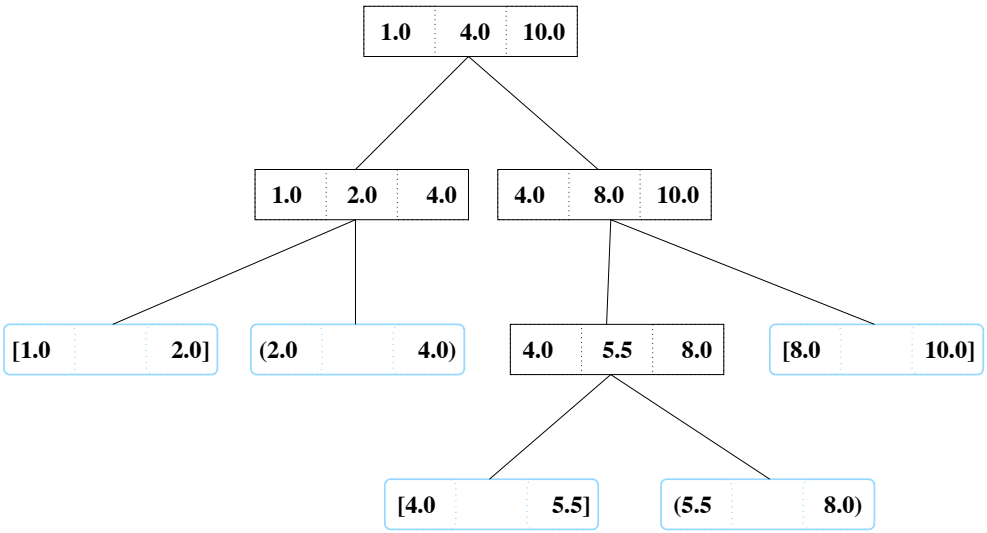

Figure 3.11: Finalized tree.

are 2.0 and 8.0. These would split the dataset into the subranges [1.0:2.0] [2.0:4.0] [4.0:10.0] and [1.0:4.0] [4.0:8.0] [8.0:10.0] respectively. The gain in log-likelihood for each of the two possible divisions is written in the half-circle over the not-yet-exercised cuts. The cut at 2.0 results in a log-likelihood gain of 38.5 computed based on Equation 3.2, the cut at 8.0 has a log-likelihood gain of 44.6. The state of the priority queue is:

```
Priority Queue:
 1. [4.0 10.0]  cut at 8.0  log-likelihood gain = 44.6
 2. [1.0  4.0)  cut at 2.0  log-likelihood gain = 38.5
```

Among the possible cuts the one with the largest gain in log-likelihood is selected, which in this case is the cut at 8.0, and the split is performed. Figure 3.10 shows the state of the discretization tree after two cuts. With the

cut at 8.0, two new bins are generated, and in each of them a new possible cut is searched for. These cuts are 5.5 and 9.9, with log-likelihood gains of 100.3 and 9.5 respectively. So, for the third cut, there is a choice between three cuts (including the cut at 2.0). The state of the priority queue is as follows, with the possible cut at 5.5 at the top of the queue.

```
Priority Queue:
 1. [4.0   8.0]  cut at 5.5  log-likelihood gain = 100.3
 2. [1.0   4.0)  cut at 2.0  log-likelihood gain = 38.5
 3. (8.0  10.0]  cut at 9.9  log-likelihood gain = 9.5
```

Based on this, 5.5 is chosen as the next cut point. After four cuts the tree learning algorithm decides to stop based on using cross-validated likelihood as the stopping criterion. Note that for this criterion, ten fully-grown auxiliary trees are constructed.The cross-validated log-likelihood curve attains the maximum at four cut points and therefore four cuts are performed and the algorithm does not add any further nodes.

Figure 3.11 shows the final tree. The resulting histogram is shown in Figure 3.8. In the final tree each leaf node represents a bin of the histogram. Each internal node represents a cut.

3.8 The Problem of Narrow Cuts

On some datasets the split into a subrange can show the problematic result of a very narrow cut. This happens when instances at one end of a range to be split lie very closely together. The log-likelihood criterion is unstable at the ends of a range and the algorithm has the tendency to cut off these few points. The number of instances in the resulting bin can be very small, perhaps containing only two points. If these cuts happen, it results in an estimate that does not reflect the true underlying density function and this also distorts the histogram (see Figure 3.12).

To avoid these narrow cuts, TUBE uses a heuristic approach. Two heuristics have been developed: I. Disallowing cuts that are very small and have very few instances relative to the dataset size and the range's width; II. Setting a minimal bin width that is derived from a cross-validated equal-width binning performed on the same sample.

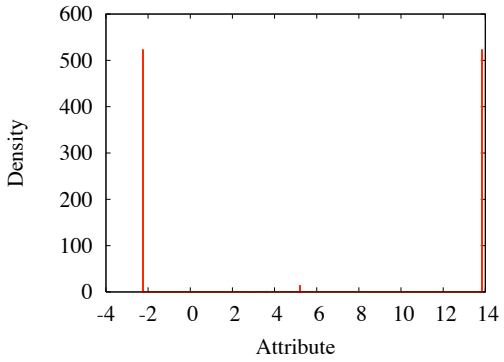

Figure 3.12: Distorted histogram due to small cuts.

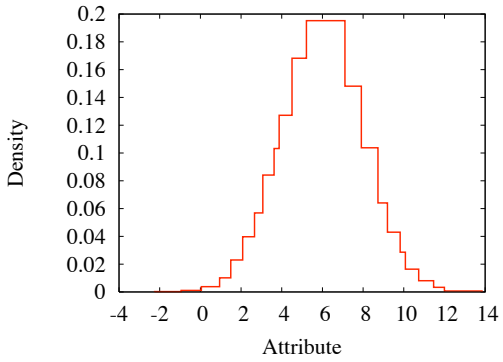

Figure 3.13: Small cuts eliminated with heuristic.

3.8.1 Heuristic I - Based on Dataset Size

The first heuristic is based on the training dataset size and the width of the total range. More specifically, the heuristic disallows cuts that are smaller than 0.1 percent of the whole range of the data and sets the minimum number of instances to $\sqrt{0.1 \times N}$ where N is the total number of instances in the dataset. This value was chosen according to a similar heuristic used in the equal-frequency discretization method PKID [70] where the number of instances per bin is set to $\sqrt{0.1 \times N}$. For a more detailed explanation of PKID see Section 4.2.2.

Figure 3.12 shows a strongly distorted histogram estimate of a normal density that is due to two small cuts that have very high density. This was created by the tree learning algorithm without using the heuristic. The same dataset is used in Figure 3.13, where the small cuts have been avoided using the heuristic and the resulting density estimate has the desired shape.

3.8.2 Heuristic II - Based on EW-Cross-validated

In experiments with the method of estimating densities using cross-validated equal-width histograms, it was observed that the resulting histogram can very precisely model large changes in the underlying density, but does not react to 'spikes' in the data's distribution in the same way as TUBE does. To combine the best properties of both approaches, TUBE's second heuristic for avoiding narrow cuts is based on taking half of the bin width from the cross-validated equal-width histogram estimate as minimal bin width.

3.9 Comparing TUBE with Standard Binning Methods

TUBE's binning can be compared with the standard histogram estimate obtained using equal-width and equal-frequency binning. This comparison will be done in detail in the next chapter (Chapter 4) with the application of TUBE to discretization.

TUBE, as well as the equal-width and equal-frequency methods, can be used as a discretization method. Discretization is a preprocessing method in machine learning that is used to change a continuous attribute into a categorical attribute. It is frequently used whenever a method cannot work with a continuous variable but only with a categorical one.

TUBE's potential advantage in this context is that it can find strong discontinuities like empty spaces and 'spikes' in the data, where many instances have very similar values, and that it shows these discontinuities clearly in the resulting histogram. It will be important to see how TUBE can compete with these very simple methods concerning runtime.

3.10 Empirical Evaluation Using Naive Bayes

To extend the empirical evaluation of TUBE, the method is also applied to naive Bayes classification. The next chapter (Chapter 4) will discuss the application of the univariate TUBE method in both areas, discretization and naive Bayes classification. In the discretization case, TUBE will be compared with other discretization methods using the likelihood of the binning as the evaluation criterion. For the second application, TUBE is used as a density estimator inside naive Bayes. It is compared with naive Bayes combined with other, standard density estimation methods. For this evaluation, a standard criterion for evaluating classification methods, namely the classification accuracy is used.

3.11 Summary

This chapter has explained the technical details of the univariate TUBE binning method. TUBE's binning results in a histogram. One challenge when designing tree-based algorithms is to have a good heuristic for choosing the

next cut. This was solved in TUBE by using the log-likelihood gain as criterion. A greedy algorithm is applied to choose the next node to be expanded. The split in a subrange is also selected using the log-likelihood criterion. To avoid problems with real-world data, cut points are always set a fixed distance from the actual data values. This fixed distance can be changed to a different value via user parameter.

Ten-fold cross-validation is used to implement a stopping criterion for the splitting process, again using the log-likelihood of the binning as score function. A second local stopping criterion is also implemented, which uses the MDL principle to decide if a range is continued to be split.

The construction of a density estimation tree can be compared to the construction of a decision tree. Breiman et al. [9] list three elements that comprise the tree construction process for the classification case: 1."The selection of the splits", 2. "The decision when to declare a node terminal or to continue splitting it", 3. "The assignment of each terminal node to a class". This is aligned well with the parts of TUBE's tree construction process for density estimation. For 1., the TUBE algorithm decides where to split a subrange so as to adapt the histogram well to the distribution of the sample data. For 2., TUBE uses two pruning methods for the control of the tree size: a local method which continues splitting if the log-likelihood improvement is larger than a certain threshold, and a global method using cross-validated log-likelihood. For 3., TUBE is an unsupervised method but can be applied to density estimation when a density value is assigned to each bin.

Empty bins are problematic when the method is applied to test data to compute an estimated log-likelihood. To overcome the problem with empty bins, the TUBE estimator notionally spreads one instance over the entire range of the histogram. Another difficult problem concerns the case where very few instances lie very close together and the split criterion decides to cut them off. TUBE uses heuristics to avoid the distortion of the density estimate due to these narrow cuts.

In the next chapter, the univariate TUBE method will be applied in two areas. As discretization method, it will be compared to other discretization methods using the log-likelihood. The comparison will also demonstrate further properties of the method. It will also be applied as density estimator in the naive Bayes classification method, and compared with naive Bayes used in conjunction with other density estimation methods. The two applications provide an empirical evaluation of the quality of the histograms generated by

the univariate TUBE algorithm.

The TUBE method can also be extended for the multivariate case. Two later chapters in this thesis (Chapters 5 and 6) explain the implementation details of the multivariate TUBE binning process and provide applications of it.

Chapter 4

Applications of Univariate Density Estimation Trees

This chapter discusses the evaluation of the univariate TUBE method in two applications. First, as a discretization method, which is compared to other unsupervised discretization methods, based on the quality of the density estimate that they generate. The target application in this case is the generation of high-quality histograms. The second application also uses TUBE as a discretization method but applies it to density estimation within the classification algorithm naive Bayes.

TUBE's univariate binning provides the thresholds for discretization. Using these thresholds, continuous data can be transformed into discrete data. This data preprocessing step is necessary whenever a machine learning method can only handle discrete input data or the transformation from continuous to discrete data gives an improvement for the methods applied to it. However, it can also be used to build a density estimation model with the thresholds defining the bins of the histogram.

Most discretization methods have been designed for application in classification methods. As a result, several supervised discretization methods have been developed but only few unsupervised ones. TUBE discretization is an unsupervised method, it does not take the class value into account when selecting the subranges.

Both applications of univariate TUBE discussed in this chapter apply the discretization generated by TUBE to density estimation. There exists a direct connection between density estimation using binning and discretization. If the discretization is defined by hard boundaries, one can build a histogram—which is a standard density estimator—from this discretization by taking each

subinterval that was formed and taking the density as the height of the corresponding bin. Vice versa, binning density estimators define bins with hard or soft boundaries and these boundaries can be used as thresholds for discretization. Hence these methods can be viewed as discretization methods as well as density estimation methods.

In the first application, TUBE discretization is compared to other unsupervised discretization methods used as histograms. For the evaluation of the techniques in this comparison the cross-validated log-likelihood criterion is used. The second application evaluates TUBE's discretization for density estimation inside the classification method naive Bayes. Here, performance is measured using standard classification error estimates.

4.1 Application: Discretization for Histogram Estimation

Discretization means the range of a numeric attribute is divided into mutually exclusive intervals. Univariate TUBE does exactly this and therefore TUBE can directly be applied as a discretization method. The boundaries of TUBE's resulting bins are the thresholds for discretization. Like most discretization algorithms, TUBE performs univariate discretization. Multivariate discretization considers the values of more than one attribute when splitting the range.

The reasons why discretization is performed on data are various. For some machine learning algorithms the runtime is correlated with the number of different values of the attribute. Continuous data can often have many different values. To reduce the number of these values, the range can be discretized into subintervals and all values falling into a subinterval are substituted by the mean or the median of this interval. This transformation does not change the attribute's type and it stays numeric.

A machine learning algorithm that cannot handle continuous data requires a preprocessing step to transform numeric attributes into categorical ones. The range of each attribute is discretized into subintervals and all attributes' values are substituted by an identifier, also called a label, for each interval. The resulting attribute has an order defined on its labels. The ordered categorical data type is also referred to as *ordinal* [21]. Frank & Witten [21] point out that a numeric attribute simplified to an ordered attribute can have the effect that a classification model built from the transformed data is less complicated and overfitting is avoided.

Discretization methods differ in the way they select the thresholds between subintervals. Equal-width discretization splits the area into subintervals of equal length. Equal-frequency discretization chooses the subintervals so that approximately the same number of values falls into each interval. On the other hand, TUBE discretization utilizes the log-likelihood criterion to find the next split point. It tries to find subintervals that represent areas of uniform density in the density function.

The decision determining how many intervals are formed for a range either has to be made by the user or is automatically computed by the algorithm. In contrast to the other two basic methods, TUBE finds the number of intervals automatically based on the cross-validated log-likelihood.

There are different ways to interpret the thresholds. Apart from the stan-

dard 'hard' thresholds, they can also be defined as 'soft', with an area around the boundaries of two adjacent subintervals where the values can be part of both the left and the right interval. 'Soft' boundaries can be defined using probability functions [59]. Like the other two discretization methods mentioned above, TUBE is a discretization method with 'hard' thresholds.

In the literature, discretization is mostly discussed in the context of classification methods and there are several supervised discretization methods that successfully improve prediction performance. Supervised methods take the class attribute of the instances into account when discretizing a continuous attribute.

In contrast, TUBE discretization is an unsupervised method. For evaluation of the method in the context of histogram estimation, TUBE will be compared to other univariate, unsupervised discretization methods. Section 4.1.3 contains the list of methods that are used for comparison.

Most discretization methods are splitting or top-down algorithms [24][40] and only a few are merging or bottom-up algorithms. TUBE's top-down algorithm for generating histograms is explained in Chapter 3.

Splitting of a numeric range can also be a useful transformation of data when the goal is to improve the ability to gain knowledge from the data [24]. For instance, the values 'high', 'medium' and 'low' can potentially be more easily interpretable than the numeric values themselves. TUBE's discretization selects its subintervals by looking for areas of near uniform density, which is not always an effective criterion for finding good sub-concept areas. Section 4.1.2 gives a few examples and discusses aspects of interpretability.

To summarise the structure of this section: Section 4.1.1 introduces several well-known discretization methods that constitute related work and discusses their main aspects. The topic of discretization for interpretability is briefly discussed in Section 4.1.2. Section 4.1.3 contains the details of the empirical evaluation of TUBE discretization, where it is compared to other unsupervised univariate methods. This evaluation is followed up by a section (Section 4.1.4) about a visual evaluation and comparison of the TUBE histograms using a selection of examples. Section 4.1.5 discusses a short experiment with varying cut distance on some of the same datasets. Findings from the application of TUBE to discretization is summarised in Section 4.1.6.

4.1.1 Related Work

Liu et al. [40] give an overview of discretization methods and define a typology that splits them into supervised and unsupervised, dynamic and static, and splitting and merging methods.

The difference between supervised and unsupervised discretization was explained above. Dynamic and static methods so far only exist as properties of supervised algorithms. Dynamic discretization is interwoven with classification activities like the building of a classification tree. Static methods finish with the discretization before the classification is started. TUBE is applied as a static discretization method. Further definitions used are local and global discretization (e.g. by Dougherty [15]). This is similar to the distinction between dynamic and static methods.

TUBE is a top-down algorithm like most existing discretization methods. The ChiMerge algorithm [34] is a bottom-up algorithms developed for supervised discretization.

A further typological characteristic is whether the algorithm performs univariate or multivariate discretization. Multivariate discretization considers more than one attribute when splitting. The TUBE algorithm as applied in this chapter performs univariate discretization.

Unsupervised Discretization

TUBE is unsupervised and is compared in this thesis to equal-width and equal-frequency discretization, which are also unsupervised (see also Section 4.1.3 for the list of methods TUBE is compared to). Both methods normally require the number of bins to be set by the user. However, Yang & Webb [69] improve unsupervised equal-frequency discretization in the context of using it for the naive Bayes classifier by computing the number of intervals from the given number of training instances by setting the number of intervals to $\sqrt{N}$. The number of instances in each interval is thus approximately $\sqrt{N}$. Hence their method like TUBE, does not need user input for this parameter.

Supervised Discretization

In the machine learning literature, discretization is mainly discussed as a preprocessing step for classification tasks. For this task, supervised methods generally produce better results than unsupervised ones.

Holte [29] developed 1RD (i.e. one rule discretizer), a simple discretizer for his 1R (i.e. one rule) decision tree classifier. The bins are formed by cutting with the aim of having instances of one class only in each bin. To avoid too many small bins, a minimal number of instances in a bin is set.

In the decision tree classifier 1RD [29], numeric attributes are dynamically discretized. At each selection of a new tree node, all continuous attributes are sorted anew and between each two consecutive values a possible split value is computed. Later algorithms tried to simplify and improve this algorithm. Catlett developed the supervised method D-2 [10], which cuts an attribute independently of the other attributes by increasing the information gain on the classification criterion. It works in a static fashion and saves computation time since the attribute is not resorted in each node as it is done in 1RD. Pfahringer [49] uses the MDL principle to decide which cut points to use. The cut points are first generated with D-2. His global splitting procedure allows a decision tree algorithm to perform multi-way splits on attributes.

Fayyad & Irani's [18] discretization method is entropy based and also static. It is already explained in a previous chapter (see Chapter 2, Section 2.3.1). Kohavi & Sahami [37] compare error-based and entropy-based discretization and find that entropy-based discretization prior to decision tree construction results in more accurate models than error-based discretization.

Kerber's ChiMerge discretization method [34] starts with each value in one interval and merges intervals in a bottom-up fashion using the statistical χ^2 test. The stopping criterion depends on a user parameter α, which sets the threshold at a selected significance level.

Multivariate Discretization

Univariate discretization only considers the attribute to be discretized, not the values of other attributes. Multivariate discretization takes more than one attribute into account for selecting the intervals. Univariate discretization can not detect XOR-like patterns in the data.

Bay [7] implements multivariate discretization as an application for set mining. Set mining is aimed at finding new insightful patterns in data. His Multivariate Discretization (MVD) algorithm splits each attribute in a fine-grained fashion and then combines neighbouring bins of similar density.

Although this thesis only evaluates TUBE in the context of univariate discretization, the multivariate TUBE algorithm Multi-TUBE (see Chapter 5) could be used for multivariate discretization. Multi-TUBE finds multidimen-

sional rectangular areas of similar density, which could be used as input for Bay's MVD algorithm.

4.1.2 Interpretable Intervals

This subsection explores TUBE's ability to form interpretable intervals based on two examples. Note that what makes an interval interpretable is not easy to define. Hence, this aspect of discretization is not fully explored. This subsection will only provide a brief inspection of the problem.

Example: Attribute with Age Values

In the first example, an attribute containing the age of a group of people is split into subintervals. The attribute used is the age attribute of the well-known diabetes dataset (diabetes-8) from the UCI dataset collection [6]. The people in this dataset are of age 21 and older, with the oldest person in the sample being 81 years old. Figure 4.1 shows the histogram built by TUBE when selecting the number of bins automatically using cross-validation. The algorithm chooses ten intervals, which seems too many for age data to suit as an easily interpretable partition.

Intuitive partitioning would perhaps select a smaller number of bins. The intuitive partitioning procedure introduced in [24] (as the *3-4-5 rule*) splits the range of values into three to five intervals of equal-width. In accordance with this rule, another histogram can be built with TUBE by setting the number of intervals to three via TUBE's user parameter. Note that, as usual, TUBE selects bins of varying width.

Figure 4.2 shows the resulting histogram. The interval boundaries for this 3-bin histogram are 28.5 and 46.5. A more natural partitioning would require the thresholds to have round values like 30.0 and 50.0. However, because TUBE cuts a range, between two values of the corresponding ordered training set, the thresholds rarely are 'round' numbers. Nevertheless, the split into these three subintervals— 'young', 'middle age' and 'old'—seems reasonable. However, it is possible that a more intuitive discretization for the domain expert exists that depends on other attributes of the datasets (like blood pressure, heart rate etc.). This kind of discretization cannot be found using univariate TUBE, which only considers the attribute itself.

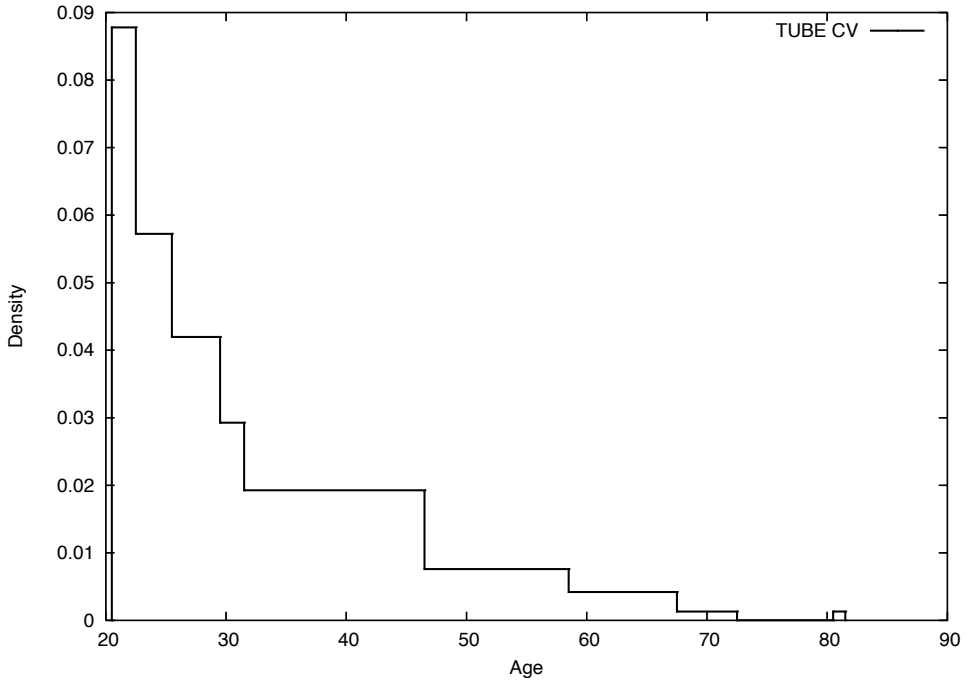

Figure 4.1: TUBE discretization for the age attribute in diabetes data. TUBE finds ten intervals.

Example: Extracting Areas with Outliers Only

As a further example, consider a discretization obtained from 500 values, which exhibit a Gaussian distribution with some additional noise. The intuitive splitting, as mentioned before, selects intervals of equal-width and therefore outliers can destroy the value of the partitioning. In this case TUBE can be used in a preprocessing step to find low density areas corresponding to outliers.

Figure 4.3 shows how TUBE discretization splits the range into nine intervals and finds several areas with low density. To show these areas more clearly, Figure 4.4 zooms into the lower density values. Depending on the application concerned, these bins may contain outliers and noise only. Using a user-given threshold, e.g. 0.01, for the minimal density value, these areas can be excluded before intuitive splitting is applied on the remaining range.

4.1.3 Empirical Evaluation of TUBE Discretization

The experiments performed in this subsection evaluate how well TUBE discretization estimates the true density. The density estimates that are generated are evaluated using 10×10-fold cross-validation, measuring the log-likelihood on the test data. TUBE is compared to several other unsupervised discretiza-

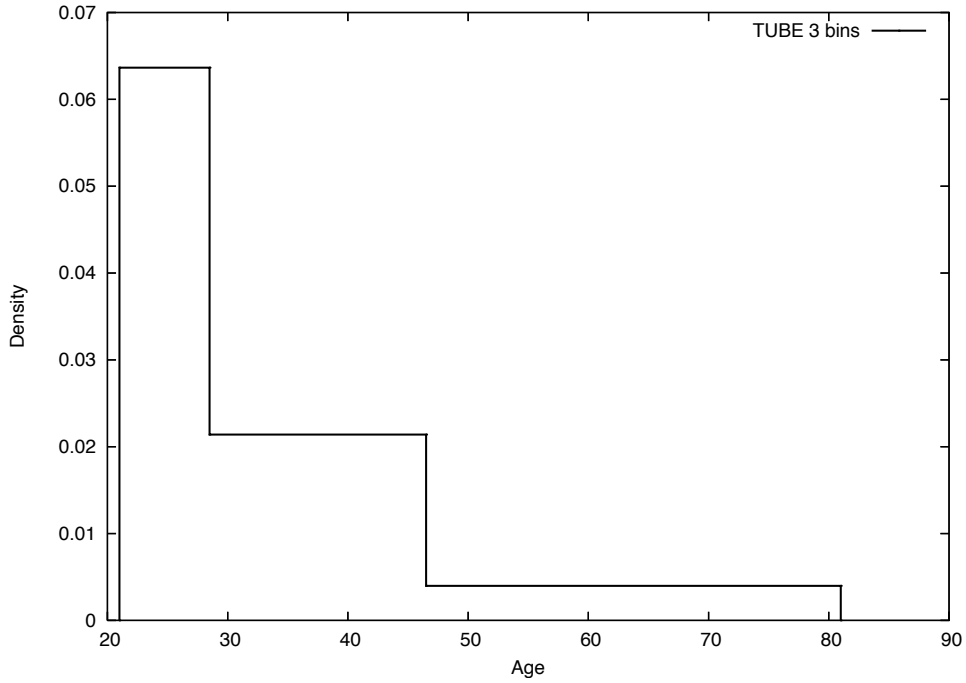

Figure 4.2: TUBE discretization of age attribute with three subintervals.

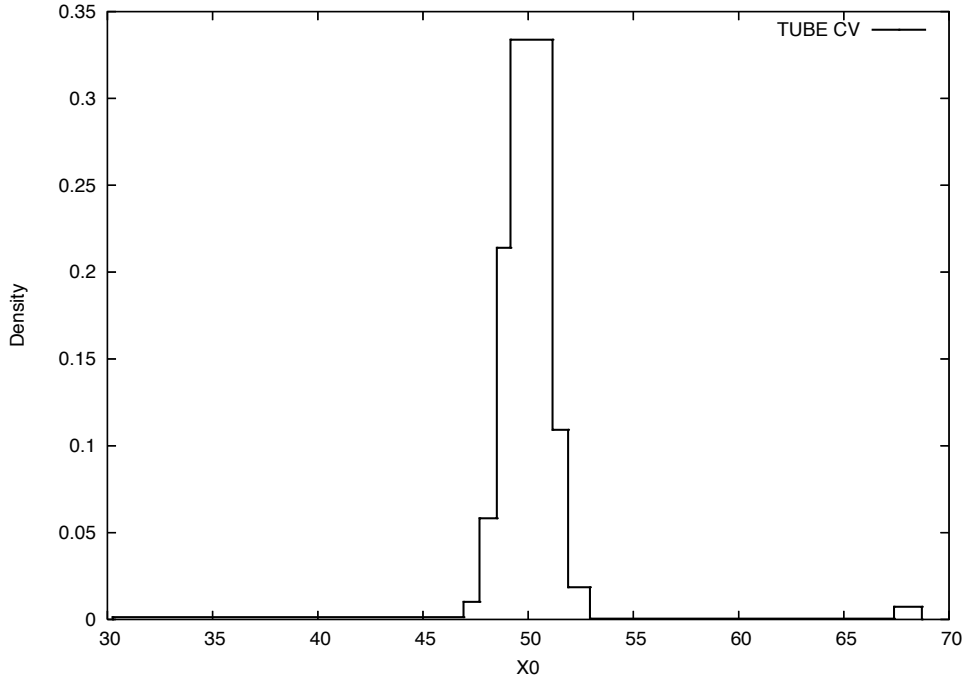

Figure 4.3: TUBE finds the low density areas.

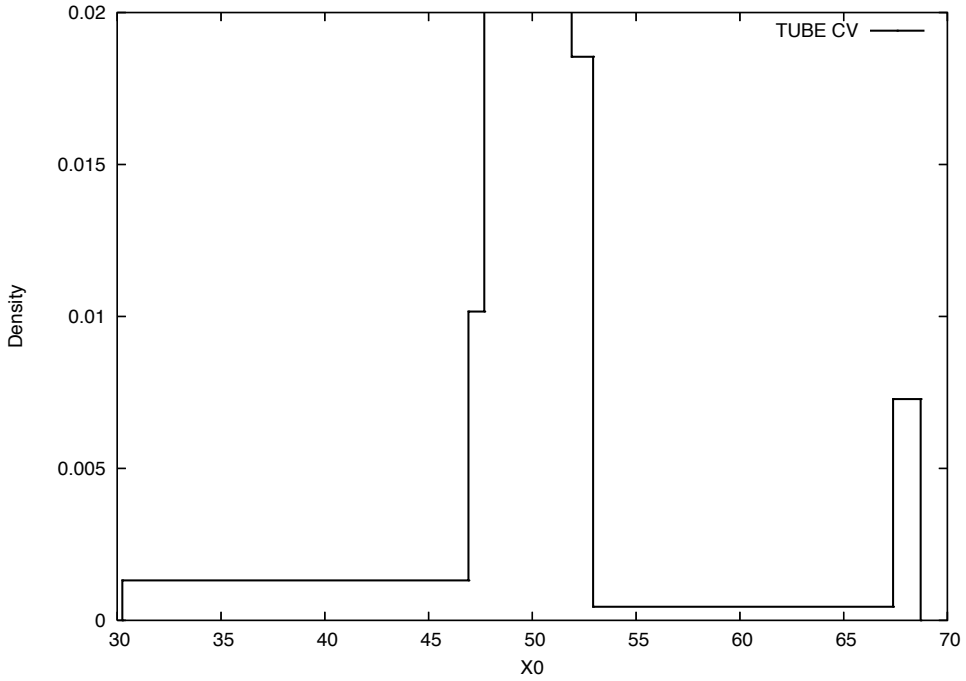

Figure 4.4: TUBE discretization, zoomed in on the low values.

tion techniques. Cross-validation is a standard evaluation method used in machine learning. Chapter 2, Section 2.1.5 discusses cross-validation in detail.

Criterion for Empirical Evaluation

The log-likelihood is a commonly used measure to evaluate density estimators [60] and is a measure for how likely the model is, given the data. The log-likelihood criterion is already discussed in previous chapters: Section 2.1.5 covers evaluation of density estimation in general, and Section 3.2 explains the way TUBE itself decides between models using the log-likelihood. The experiments presented here use the same log-likelihood measure to compare TUBE discretization with other discretization methods. In the case of TUBE, an 'inner' cross-validation is applied on each training fold of the 'outer' cross-validation to obtain an appropriate number of intervals.

Repeated here for convenience, Equation 4.1 shows the log-likelihood computed from a test dataset that tests the model built from the corresponding training dataset. Let n_i be the number of training instances in bin i, $n_{i_{test}}$ be the number of instances of the test set that fall into this bin, w_i be the bin width, and N be the total number of training instances. Then the log-likelihood L on the test data is:

Table 4.1: 464 numeric attributes from UCI datasets and their levels of uniqueness.

Dataset	[0-20)	[20-40)	[40-60)	[60-80)	[80-100]	num inst
anneal	6	-	-	-	-	898
arrythmia	182	7	14	3	-	452
autos	13	-	-	1	1	205
balance-scale	4	-	-	-	-	625
winsconsin-breast-cancer	9	-	-	-	-	699
horse-colic	7	-	-	-	-	368
german-credit	6	-	-	-	1	1000
ecoli	7	-	-	-	-	336
glass	3	3	2	1	-	214
heart-statlog	12	1	-	-	-	270
hepatitis	4	1	1	-	-	155
hypothyroid	7	-	-	-	-	3772
ionosphere	2	-	2	31	-	351
iris	4	-	-	-	-	150
labor	8	-	-	-	-	57
lymphography	3	-	-	-	-	148
segment	14	3	-	2	-	2310
sick	7	-	-	-	-	3772
sonar	-	7	4	-	46	208
vehicle	17	1		5	-	846
vowel	-	-	4	8	-	990
Sum	315	23	27	51	48	
In percent	68	5	6	11	10	

$$L = \sum_i n_{i_{test}} \times log \frac{n_i}{w_i \times N} \tag{4.1}$$

In the case of cross-validated equal-width discretization, which is one of the benchmark methods TUBE is compared to, leave-one-out cross-validation can be applied as the 'inner' cross-validation because the log-likelihood for each test instance can be easily computed due to the fact that the bins stay fixed. In TUBE discretization the location of each cut point can change with one instance removed, making the leave-one-out method too expensive. Therefore 10-fold cross-validation is used instead.

Datasets Used for Evaluation

The TUBE discretization method is evaluated using numeric attributes from 21 UCI datasets [6]. The algorithm works on univariate numeric data, and thus the numeric attributes of the UCI datasets have been extracted and converted into 464 one-attribute datasets.

A surprising finding was that many of these numeric attributes have a

low uniqueness in their values. Low uniqueness means that they have many instances with the same value. Table 4.1 lists the number of attributes sorted into columns according to their level of uniqueness (e.g. '$[0-20)$') means that the percentage of unique values is between 0 and 20). The table also shows the UCI datasets the attributes have been extracted from and the number of instances.

The low uniqueness values can be explained by the high percentage of discrete attributes in the data. More than 50 percent of the attributes have integer values only and several others have only few different values because of low precision measurements. The attributes from the ionosphere dataset represent a special case of low uniqueness because the values are radar returns. The nature of the radar results in many values at 0.0, 1.0 and -1.0, but continuous data between these values.

Discretization Methods Compared to TUBE

TUBE discretization is compared to equal-width and equal-frequency discretization, methods, which are explained in detail in Chapter 2 as equal-width and equal-frequency histogram respectively.

TUBE is compared against equal-width discretization with ten bins (EW-10), equal-width discretization with cross-validation for the number of bins (EWcvB), equal-width discretization with cross-validation for the origin of the bins and the number of bins (EWcvBO), and equal-frequency discretization with ten bins (EF-10). Note that the equal-frequency method could not produce useful models for attributes with uniqueness lower than 20 and has therefore been left out of the comparison for those cases. TUBE, EWcvB and EWcvBO were all run with the maximum bin number set to 100. All methods were implemented in the WEKA machine learning software [68].

Experiments

Table 4.2 provides a summary of the comparison. Each value in the table is the percentage of all attributes in that uniqueness category for which TUBE was significantly better, equal or worse respectively, based on the corrected resampled t-test [47]. In almost all cases TUBE is at least as good as the other methods and produces especially good results in cases with low uniqueness and some cases of high uniqueness. An analysis of the corresponding attributes shows that TUBE is generally better when attributes exhibit discontinuities

Table 4.2: Comparison of the density estimation results. Result of paired t-test based on cross-validated log-likelihood.

	EW-10	EWcvB	EWcvBO	EF-10
(0-20)				
TUBE significantly better	99	100	100	-
TUBE equal	1	0	0	-
TUBE significantly worse	0	0	0	-
[20-40)				
TUBE significantly better	48	43	43	48
TUBE equal	52	57	57	52
TUBE significantly worse	0	0	0	0
[40-60)				
TUBE significantly better	8	8	8	37
TUBE equal	92	92	92	63
TUBE significantly worse	0	0	0	0
[60-80)				
TUBE significantly better	53	56	56	67
TUBE equal	44	40	42	30
TUBE significantly worse	3	3	2	3
[80-100]				
TUBE significantly better	13	17	15	13
TUBE equal	85	81	81	85
TUBE significantly worse	2	2	4	2
Total				
TUBE significantly better	76	77	77	43
TUBE equal	23	22	22	55
TUBE significantly worse	1	1	1	2

Table 4.3: Comparison of the number of bins.

	EW-10	EWcvB	EWcvBO	EF-10
(0-20)				
TUBE significantly fewer	14	62	62	-
TUBE equal	2	8	7	-
TUBE significantly more	84	30	31	-
[20-40)				
TUBE significantly fewer	31	13	26	31
TUBE equal	4	30	17	4
TUBE significantly more	65	57	57	65
[40-60)				
TUBE significantly fewer	29	46	54	29
TUBE equal	38	42	38	38
TUBE significantly more	33	12	8	33
[60-80)				
TUBE significantly fewer	44	94	97	44
TUBE equal	14	6	3	14
TUBE significantly more	42	0	0	42
[80-100]				
TUBE significantly fewer	96	85	92	96
TUBE equal	2	15	8	2
TUBE significantly more	2	0	0	2
Total				
TUBE significantly fewer	29	65	68	56
TUBE equal	5	12	9	12
TUBE significantly more	66	23	23	32

in their distributions.

It is difficult to split the datasets precisely into attributes with continuous distributions and attributes with discontinuous distributions. Datasets below 20 percent uniqueness can be considered discontinuous but there are some datasets in the higher uniqueness category that showed strong discontinuities.

Attributes with low uniqueness exhibit discontinuous distributions of different kinds. Some of the attributes are very discrete and have only integer values (e.g. vehicle-9) or a low precision (e.g. iris-4), some have irregularly distributed data spikes (e.g. segment-7) and some have data spikes in regular intervals (e.g. balance-scale-1). In the category of (0-20) uniqueness TUBE outperforms all other methods on almost all of the datasets.

In the category [60-80) half of the attributes have a distribution that is a mixture between continuous data and discrete data (most of the ionosphere attributes in this category have a mixed distribution). TUBE's density estimation was better for all these attributes.

Comparing the Number of Bins

Table 4.3 shows a comparison of the number of bins generated by the different methods (which is always fixed to ten for the methods not using 'inner' cross-validation). A smaller number of bins yields histograms that are generally easier to understand and analyze. In the category 80 percent and higher the TUBE discretization can adapt well to the data and generates a significantly smaller number of bins than the other methods.

4.1.4 Visual Evaluation of TUBE Discretization

Histograms are a method used for visual data exploration. Therefore it makes sense to evaluate TUBE discretization results by looking at the resulting histograms built from the discretized range. This subsection first visually evaluates the way TUBE represents certain features of a data distribution, namely empty areas and areas of varying density, and then compares it with how equal-width discretization and equal-frequency discretization can represent these features. To strengthen the argument, the size of the difference area was measured and used to compare TUBE discretization with the other methods used in this section. The difference area is the area representing the difference between the estimated density function and the 'real' function used for generating the data.

A second set of examples explores the effect of varying TUBE's parameters like the minimal bin width and the cut point distance to the data values. The resulting histograms are shown and the log-likelihood values are compared.

Discretization of Distribution with Empty Area

The first example dataset investigated contains two areas with uniform distribution and a large empty area in between them. Figure 4.5 shows the histogram built using TUBE discretization. The true generated distribution is the dotted line. TUBE's histogram and the true distribution are very close together.

Two additional histograms were generated for this dataset using simple equal-width discretization and equal-frequency discretization, both with ten bins—see Figure 4.6. The empty range is not clearly shown as in TUBE's histogram. It can be seen that the equal-frequency histogram does not identify empty areas because of the way it is constructed, with each bin containing approximately the same number of instances and never zero instances.

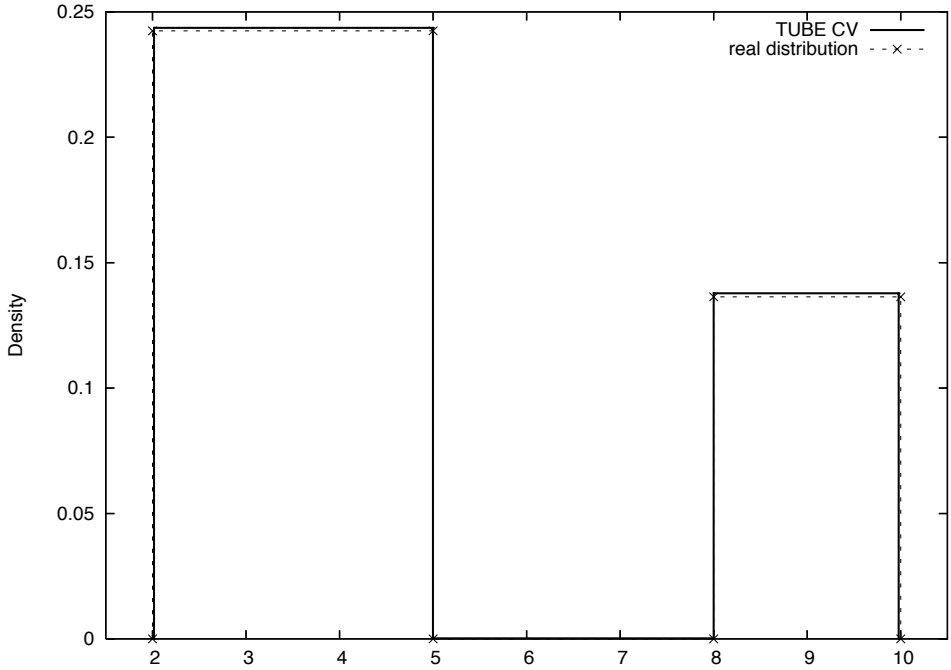

Figure 4.5: TUBE discretization: two uniform areas with an empty range.

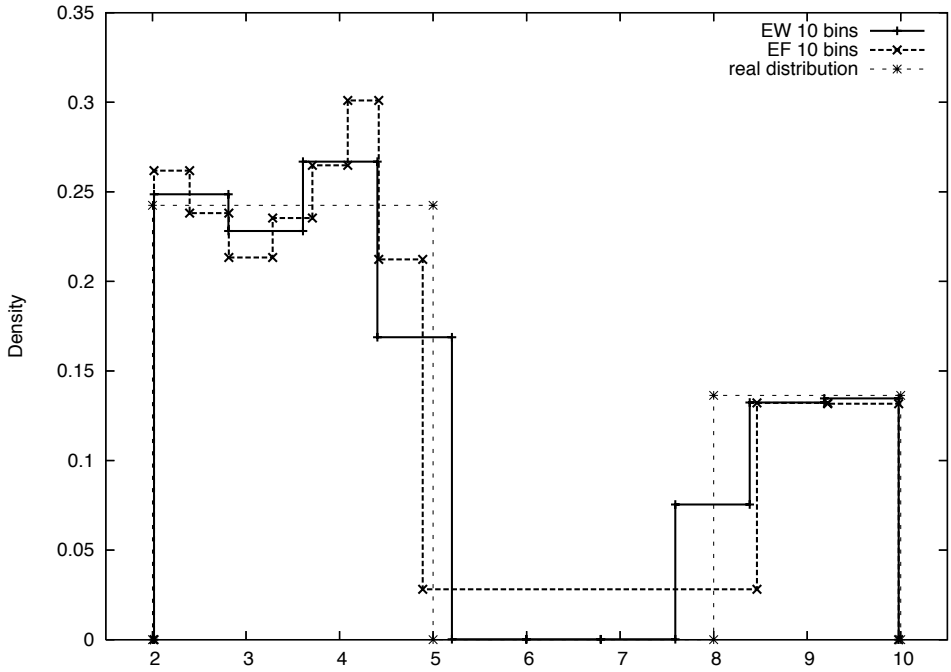

Figure 4.6: EW with ten bins and EF with ten bins: two uniform areas with an empty range.

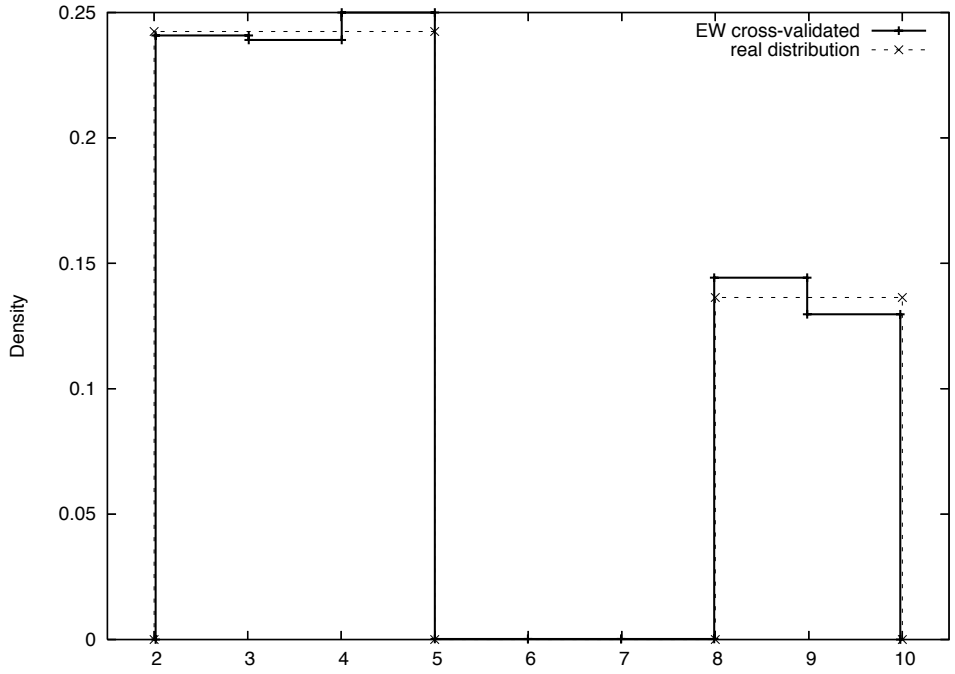

Figure 4.7: EW cross-validated: two uniform areas with an empty range.

Figure 4.7 shows a histogram built using cross-validated equal-width discretization. The algorithm finds the number of bins using cross-validation and for this dataset decides on eight bins (A similar result can be obtained by also cross-validating the origin). It defines the empty areas well, setting the bin boundaries at the points of density change, but the areas of uniform density are split into many bins instead of one as in TUBE's discretization.

Discretization of a Distribution with Extreme Changes

The dataset generated for this example has extreme changes in the distribution and consists of subranges of several different uniform densities but does not have any empty subrange like the dataset in the previous example.

Figure 4.8 shows TUBE's histogram and again TUBE fits the real distribution perfectly. A second histogram was generated with cross-validated equal-width discretization—see Figure 4.9. Cross-validated equal-width manages to adapt the number of bins and the bin origin in such a way that all big steps are defined well. However, the uniform areas get quite dissected when represented by many bins instead of one.

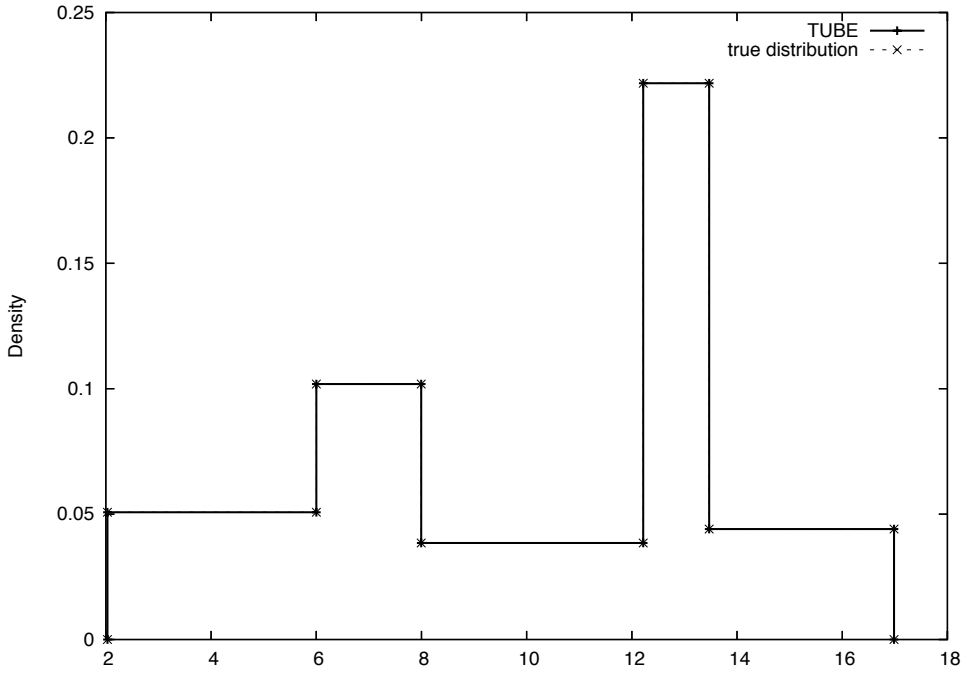

Figure 4.8: TUBE discretization: five uniform areas.

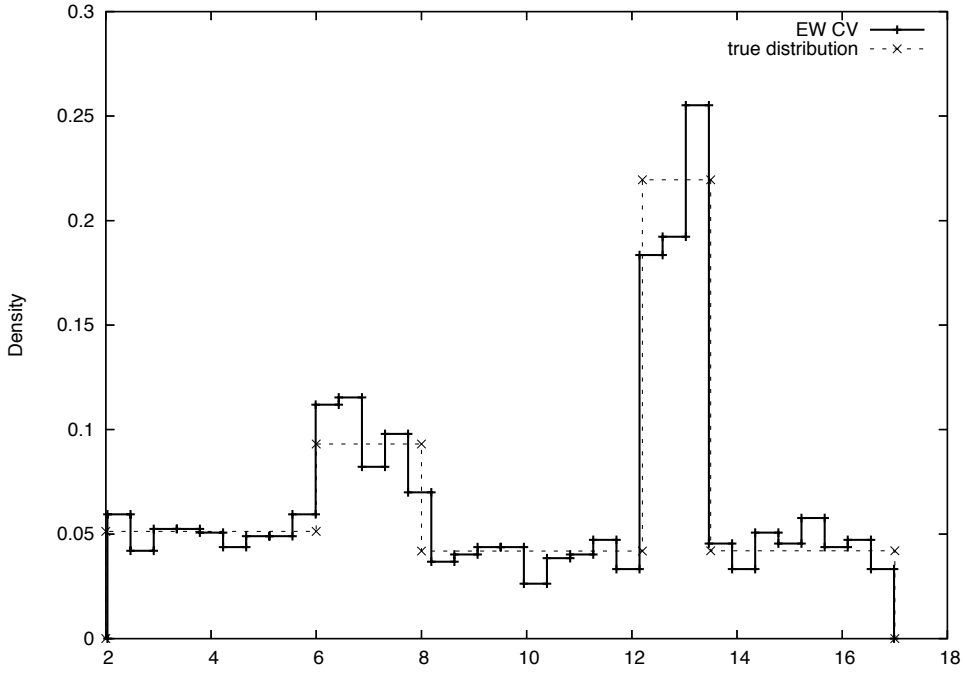

Figure 4.9: EW cross-validated: five uniform areas.

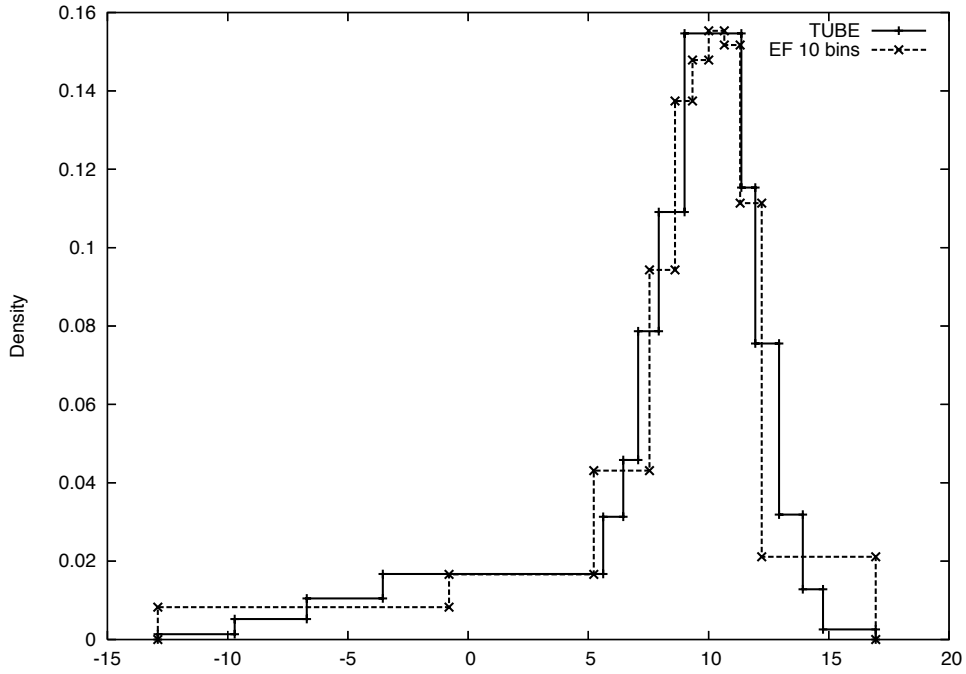

Figure 4.10: TUBE and EF ten bins discretization: two Gaussians.

Discretization of a Distribution with two Gaussians

The dataset used in this experiment contains values generated from two Gaussian distributions. Figure 4.10 shows how TUBE and equal-frequency discretization adapt to the change of density along the flanks of a Gaussian by changing the width of the bins. In contrast, the cross-validated equal-width histogram in Figure 4.11 has to select a large number of bins to represent the distribution accurately. However, it does make the two peaks discernible.

Comparison of the Discretization Methods using the Difference Area

The datasets generated for visual comparison of discretization methods ('two-uniforms', Figure 4.7; 'several-uniforms', Figures 4.8 and 4.9 and 'two-gaussians', Figures 4.10 and 4.11) are also compared using the area which represents the difference of the generated density and the modelled density function. This difference area is computed using a numerical method by splitting the range into 1000 subranges and adding all 1000 difference area sizes, with both function values measured in the middle of each subrange. With x_i being the value in the middle of the subrange i, $g(x_i)$ being the value of the generated function, $f(x_i)$ being the value of the modelled density function, and w_{1000} being the width of each of the 1000 subranges, the difference area size S is:

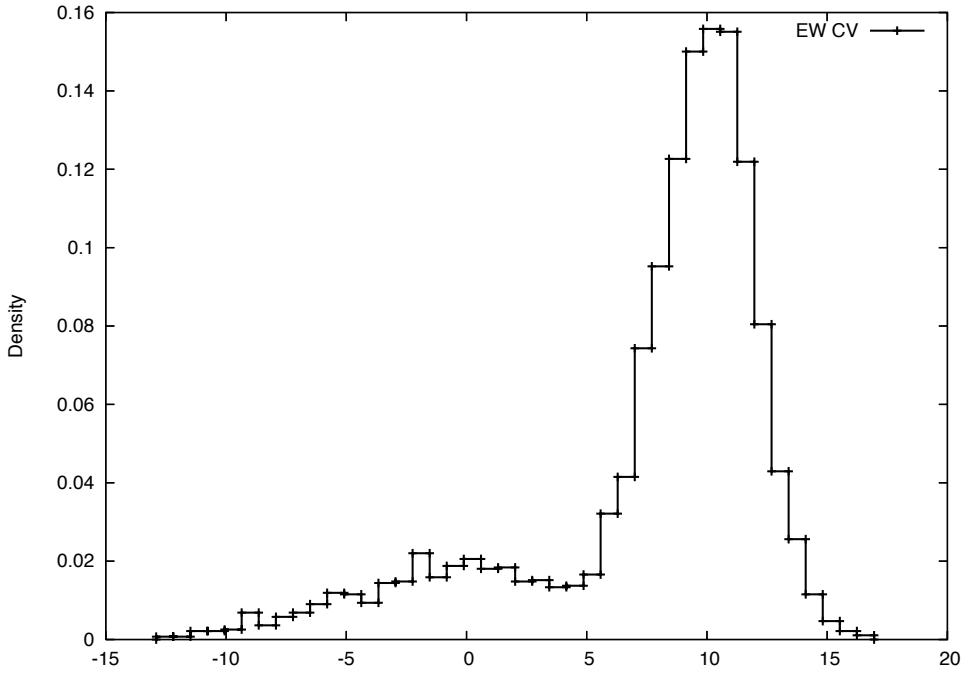

Figure 4.11: EW cross-validated discretization: two Gaussians.

$$S = \sum_{i=1}^{1000} abs(g(x_i) - f(x_i)) \times w_{1000} \tag{4.2}$$

These values were computed for the three datasets for TUBE discretization with number of bins cross-validated (TUBE), equal-width with the number of bins cross-validated (EW-cv), equal-width with ten bins (EW-10) and equal-frequency with ten bins (EF-10). For reference, values where also computed using the multi-mode Gaussian model found by the clustering method EM. (EM is explained and used for multi-dimensional clustering in Chapter 6).

The results in Table 4.4 show that TUBE discretization produces the best model for the uniform distributions and the third best for the Gaussians. EW-cv discretization produced a better fit for the Gaussians but with an unpractically high number of bins for the density model. EM's multi-Gaussian model was not unexpectedly the best for the dataset with two Gaussians. A closer look at the parameters of the two estimated Gaussian distribution shows that they are very similar to the parameters of the generated 'real' distribution (see Table 4.5).

Table 4.4: Comparing the fit of the density estimates generated by TUBE cross-validated (TUBE), EW cross-validated (EW-cv), EW with ten bins (EW-10), EF with ten bins (EF-10), and EM cross-validated (EM-cv), using the difference area size (see Equation 4.2). An asterisk ($*$) marks the smallest difference area for each dataset.

	TUBE	EW-cv	EW-10	EF-10	EM-cv
two-uniforms	0.00821 *	0.02871	0.17280	0.23360	0.17614
several-uniform	0.05262 *	0.15425	0.26731	0.14948	0.20790
two-gaussians	0.10146	0.09119	0.22770	0.24223	0.01824 *

Table 4.5: Parameters of generated Gaussians compared to those found by EM-cv.

	mean-1	stdv-1	num-1	mean-2	stdv-2	num-2
Generated Gaussian	1.00	5.0	900	10.00	2.00	3000
EM-cv	0.09	5.08	897	10.01	1.96	3003

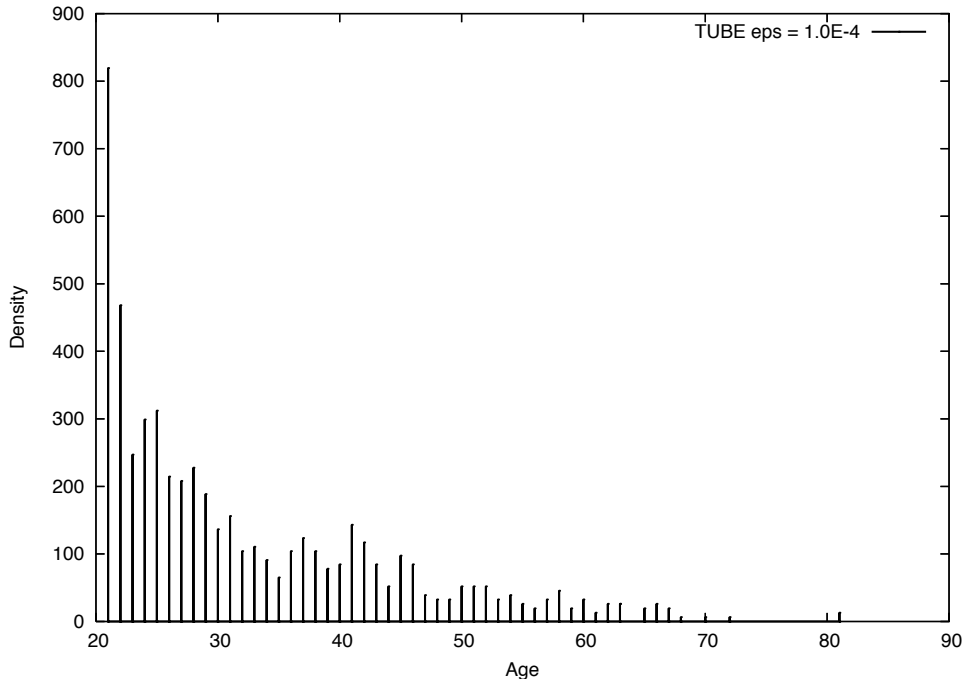

Figure 4.12: TUBE discretization of an attribute with discrete values. Cut distance is set to default $1.0E-4$.

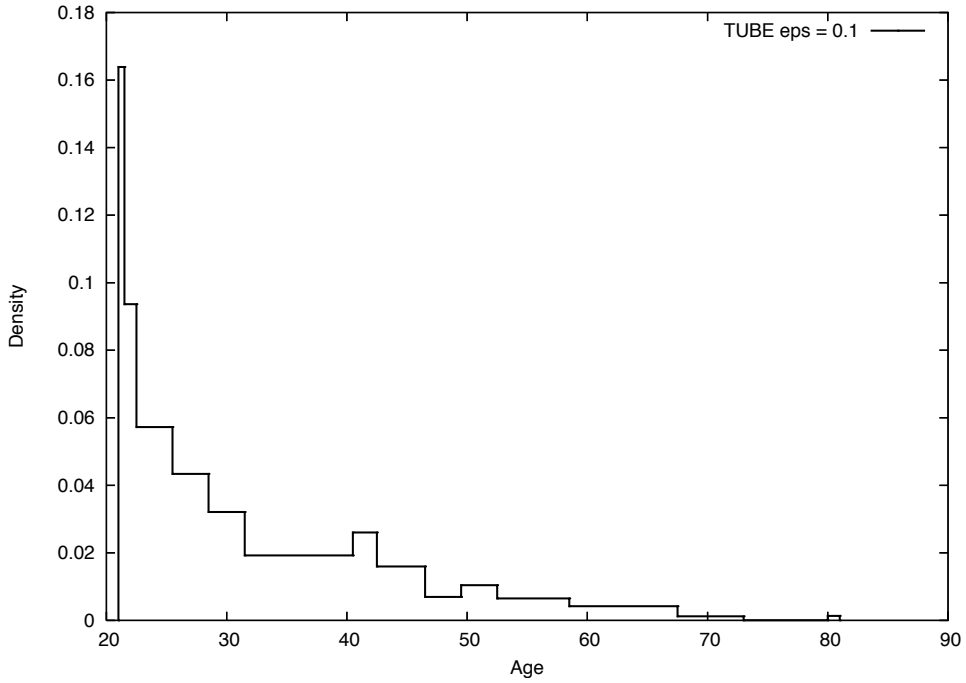

Figure 4.13: TUBE discretization of an attribute with discrete values. Cut distance is set to 0.1, larger than default.

Data with Discrete Values (Comparison of Log-likelihood)

The age attribute of the diabetes datasets (diabetes-08) is an example of a numeric attribute with discrete values where all the values are integers. TUBE's cut point distance parameter defines where splits are attempted next to each data value. For the construction of the histogram in Figure 4.12, the cut point distance (eps) was set to the default value $1.0E-4$. This setting results in a histogram that defines a bin around each discrete value and for each empty space between the values. Increasing the cut point distance to 0.1 yields a histogram that gives a more intuitive representation of the distribution of the values; see the resulting histogram in Figure 4.13. On the other hand, comparing the cross-validated log-likelihood values (10-fold cross-validation) shows that the less smoother histogram yields a higher score ($4.88 > 3.49$).

Data with Spikes based on Few Instances (Comparison of Log-likelihood)

For the construction of the histogram in Figure 4.14, based on a dataset with small spikes in the distribution, the cut point distance (eps) was set to the default value $1.0E-4$. The resulting histogram shows that TUBE finds some very narrow bins with only a few values in them. These bins form high spikes

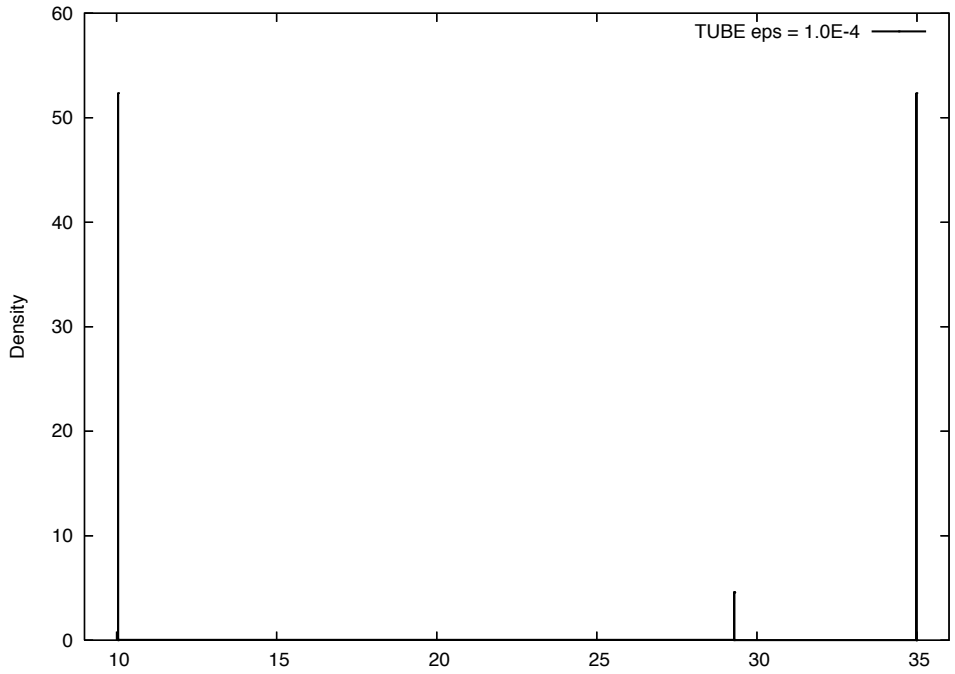

Figure 4.14: TUBE discretization of an attribute with spikes based on few values only. Cut distance is set to default $1.0E-4$.

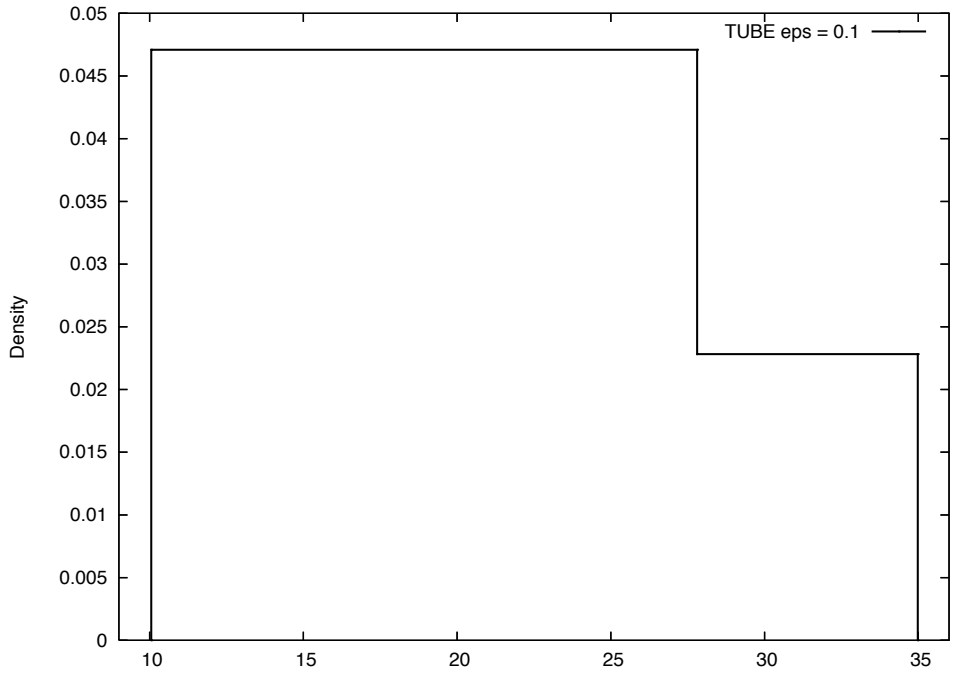

Figure 4.15: TUBE discretization of an attribute with spikes based on few values only. Cut distance is set to 0.1. The spikes have disappeared.

and render the rest of the histogram invisible. For the next histogram in Figure 4.15, the cut point distance was increased to 0.1. Moreover, bins below this distance were disallowed per user parameter. Note that, by default, TUBE cuts in the middle between two values if the cut point distance is larger than the distance of the two values.

It can be seen that with these changes to the parameters, TUBE finds a histogram which perhaps gives more useful information about the distribution of the values. However, the comparison of the cross-validated log-likelihood values (10-fold cross-validation) again shows that the less smoother histogram yields a better value ($-3.19 > -3.20$).

4.1.5 Experiments with Varying Cut Distance

To investigate the effect of the cut point distance parameter on real-world data, an experiment was performed on the UCI datasets using all the attributes in the group of attributes that exhibit [40..60) percent uniqueness. The above experiment shows that TUBE performs only sometimes better than the other discretization methods for this group. Hence, the experiments were repeated for TUBE, once leaving the setting for the cut distance (eps) at the default value ($1.0E-4$) as in the experiment before, and also setting the cut distance to the values 0.01 and $1.0E-8$.

The results are listed in Table 4.6. They show that on some attributes (glass-2, hepatitis-3, ionosphere-3) the likelihood values increase strongly when the cut distance is set to a smaller value. This is possibly because the histograms have bins which are small but have very high spikes and therefore yield very high log-likelihood values.

4.1.6 Summary of Application to Discretization

TUBE, like other unsupervised binning methods, can be used to discretize data by interpreting the bin boundaries as discretization thresholds. Discretization is often applied in data mining for preprocessing data, mostly when numeric data has to be transformed into categorical data, but it can also be used for density estimation. In this section it is evaluated whether TUBE discretization is a good algorithm for density estimation with histograms. The log-likelihood criterion and the size of the difference area was used for the empirical evaluation of artificial datasets. Moreover, some histograms were visually inspected and discussed.

Table 4.6: Results for TUBE evaluation with different values for the parameter *eps*. The values with an asterisk (*) show significant improvement based on the corrected t-test compared to the left-most value in the same row.

	eps = 0.01	eps = 1.0E-4	eps = 1.0E-8
		default	
arrhythmia-105	-1.90	-1.93	-1.71
arrhythmia-106	-2.17	-2.17	-2.17
arrhythmia-115	-2.06	-2.05	-2.10
arrhythmia-116	-2.18	-2.18	-2.18
arrhythmia-136	-1.79	-1.82	-1.75
arrhythmia-146	-2.05	-2.01	-1.97
arrhythmia-156	-2.10	-2.11	-2.11
arrhythmia-165	-2.21	-2.18	-2.16
arrhythmia-166	-2.50	-2.44	-2.34
arrhythmia-185	-2.40	-2.41	-2.40
arrhythmia-195	-2.04	-2.06	-2.13
arrhythmia-196	-2.44	-2.44	-2.45
arrhythmia-205	-1.81	-1.80	-1.82
arrhythmia-206	-2.14	-2.14	-2.11
arrhythmia-96	-1.82	-1.82	-1.84
glass-2	1.04	1.01	* 2.11
glass-7	0.75	0.71	0.98
hepatitis-3	-1.29	* 0.01	* 3.36
ionosphere-3	3.13	* 3.85	* 8.90
sonar-50	5.14	5.20	5.19
sonar-51	5.33	5.30	5.29
sonar-52	5.51	5.56	5.49
sonar-54	5.77	5.74	6.30
vowel-7	1.53	1.53	1.52

TUBE discretization produces histograms that adapt the bin size to the density changes in the distribution. It can thus represent complex distributions with sudden density changes in the data using a comparably small number of bins and can reliably detect empty spaces. On truly continuous data the method provides a discretization that represents the data as well as the other methods but with fewer bins and hence gives a clearer picture of areas of different density.

However, when the algorithm finds a range of a few instances that lie close together and are close to the border, it shows a tendency to interpret them as a feature of the distribution and decides to separate them into a bin. Also, with discrete data, the algorithm creates a bin around each value and an empty bin between two values. By modifying the minimal bin width or the cut distance these narrow spikes can be avoided.

Empirical experiments were performed on 464 numerical attributes of 21 UCI datasets [6]. The attributes have a surprisingly high percentage of discrete values. For the analysis, the attributes were split into five groups based on their level of uniqueness. The results show that, when using the log-likelihood criterion, TUBE is better or as good as the other unsupervised benchmark methods in a high percentage of cases.

It is noteworthy that TUBE, despite being a more complex algorithm than the unsupervised discretization algorithm it was compared with, did not show more than a negligible increase in runtime. On the contrary, as a greedy top-down tree-based method it is guaranteed to be practical even for large datasets.

The results show that TUBE outperforms equal-width and equal-frequency discretization on discontinuous attributes in particular. The visual investigation of those discretizations that produce well-formed histograms without spikes indicate that the histograms with spikes return higher log-likelihood values—even if the log-likelihood criterion is cross-validated—despite the fact that the spikes appear overfitted to the training data. Further experiments on the group of attributes in the $[40 - 60)$ uniqueness range, which is the range with the worst results, show that if the spikes in the dataset are made higher by reducing the cut distance parameter, the log-likelihood values get even better for some of the attributes. This is a strong indication that the log-likelihood criterion favours density estimation models which, at least visually, appear overfitted.

The next section applies TUBE discretization to density estimation for

classification in naive Bayes. The experiments for naive Bayes are evaluated using the accuracy of the classification task and provide an indirect measure of the quality of the discretization.

4.2 Application: Naive Bayes

In the application discussed in this section, TUBE discretization is used to perform density estimation inside the naive Bayes classifier. The naive Bayes classifier is a well known inductive machine learning algorithm—inductive because it uses training data to build a model to be used for the later class probability prediction for new unseen instances.

The naive Bayes classifier is a simplified Bayesian belief network. Bayesian networks use probability models of the attributes to generate a prediction. If the attribute is numeric the probabilities are generally approximated using Gaussian distributions. However, if the distributions are more complex they cannot be approximated accurately using a simple Gaussian distribution. For example, Dougherty et al. [15] find that naive Bayes classification is in most cases more accurate when the numeric attributes are discretized and transformed into categorical attributes. A different approach again is to discretize the attribute concerned and use the thresholds of the discretization to build a nonparametric density estimator. This is the approach taken for applying TUBE to naive Bayes in what follows.

Many discretization methods have been developed to support supervised discretization for decision tree algorithms. Discretization for naive Bayes does not have the same requirements as discretization for decision trees. Discretization methods developed for naive Bayes are discussed in a subsection about related work (Section 4.2.2).

Nonparametric density estimators like histograms can adapt better to complicated density distributions than parametric ones. As shown before, the TUBE discretization algorithm finds bins of varying length that adapt the estimated density function closely to the actual distribution of the training set. Note that if an attribute is simply transformed into a discrete attribute, the information about the width of the bin is lost. Hence using a density estimator instead of generating a nominal attribute through discretization should be advantageous when using naive Bayes.

To summarise the organization of this section, Section 4.2.1 explains the naive Bayes algorithm. Section 4.2.2 discusses some related work on discretization for naive Bayes. Section 4.2.3 explains the way TUBE is applied to naive Bayes. Section 4.2.4 discusses the evaluation of the application. The section ends with a summary of the application of TUBE to naive Bayes (Section 4.2.5).

4.2.1 The Classification Algorithm Naive Bayes

A Bayesian network, when used for classification, utilizes probability models of the attribute values for class probability prediction. In the case of a naive Bayesian network it is best to explain this process by starting with Bayes theorem of posterior probability. With C_i being the class of the example and X its attribute vector, Bayes' theorem says that:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)} \tag{4.3}$$

The predicted class C_i is the class for which $P(C_i|X)$ is maximal. Since $P(X)$ is the same for each class, it can be ignored because simple normalization can be applied and the following statement can be used:

$$P(C_i|X) \propto P(X|C_i)P(C_i) \tag{4.4}$$

$P(C_i)$ are the class prior probabilities. If they are not known, there are two possibilities, either all $P(C_i)$ are set to the same value or they are set relative to the class distribution in the training set. WEKA's [68] implementation of the naive Bayes classifier does the latter and is used in this way in the experiments performed for the evaluation (Section 4.2.4).

Bayes' theorem can be applied with general Bayesian networks as well as naive Bayes. The computation of the probabilities $P(X|C_i)$ can be difficult if conditional dependencies between attributes of the dataset have to be considered. In naive Bayes, these dependencies are ignored and $P(X|C_i)$ is computed using the probability of each attribute depending on the class independently. So, with n being the number of attributes and X being the tuple of attributes $(a_1, ..., a_n)$, $P(X|C_i)$, the probability of the example X depending on a class C_i, is:

$$P(X|C_i) = \prod_i P(a_i|C_i) \tag{4.5}$$

Using the training instances, the naive Bayes classifier produces probability estimates for each class and each attribute. Having a test instance X, this yields all the probability estimates required to compute the probabilities $P(C_i|X)$.

4.2.2 Related Work

In the context of the previous application, 'Discretization' (Section 4.1.1), related discretization algorithms are discussed in the general discretization setting. Discretization methods for classification have often been applied to both decision trees and naive Bayes. Yang & Webb [70] argue that naive Bayes needs special-purpose discretization methods different from those used for decision tree classification.

Dougherty [15] applies discretization as a preprocessing step to naive Bayes. In his experiments he compares equal-width discretization and the supervised methods 1RD [29] and Fayyad-Irani's entropy minimization heuristic [18]. He also compares naive Bayes with decision tree classification.

Yang & Webb [70] developed and tested their discretization method especially for naive Bayes. Their proportional k-Interval Discretization (PKID) is based on equal-frequency discretization and computes the number of intervals automatically from the number of instances by setting it to the square root of the number of instances. The complexity of their algorithm is O($nlogn$) only. PKID always keeps identical values in one interval.

Davies and Moore [12] use interpolating density trees to efficiently learn joint probability models for datasets with dozens of attributes and thousands of examples.

4.2.3 Applying Univariate TUBE in Naive Bayes

In the following, the univariate TUBE algorithm is used as a nonparametric density estimation tool to support naive Bayes classification. When building the classification models, the values of each attribute are split into subdatasets, one dataset for each class. The values for each single attribute are then ordered and TUBE performs a splitting of the corresponding range. The resulting discretization is used to generate a histogram. (See the detailed description of the univariate TUBE algorithm in Chapter 3.) Like PKID, TUBE always keeps identical values in one interval.

For the prediction of the class, the attributes $a_1, ..., a_n$ of the corresponding test example are taken to compute the probability that this example is of class C_i, using Equation 4.5. However, for continuous data, the probability $P(a_i|C_i)$ is zero. Hence, instead of the probability, the density of the histogram for class C_i and attribute a_i is used, which is generated by applying TUBE to the subdatasets that have been collected and under the presumption that this

density is strongly correlated with the probability. As mentioned above, the prior probability of class $P(C_i)$ is estimated as the proportion of the instances of this class in the training data. The class probability of the test instance $P(C_i|X)$ is computed using Equation 4.3. The class for which $P(C_i|X)$ is largest is predicted as the class for the test instance X.

For the experiments, TUBE discretization was used both with default parameter settings and with modified parameter settings that change the cut distance and the minimal bin width. The experiments are explained in detail in the next section.

4.2.4 Evaluation

Naive Bayes was tested using 21 UCI datasets [6], with all nominal attributes removed. This was done to isolate the effect of the discretization, which is obviously only applied to numeric attributes.

Several experiments were performed comparing naive Bayes using Gaussian distributions (Gauss) with either TUBE as density estimator or equal-width histograms. The first experiment used the following density estimators: Gaussians (Gauss), TUBE histogram cross-validated for the number of bins (TUBE-CV) and equal-width discretization cross-validated for the number of bins (EW-CV). To verify the usefulness of any of these classification methods, they are compared with ZeroR, a classifier that simply predicts the majority class. All experiments were performed using 10×10 cross-validation and all comparisons are based on the corrected resampled t-test [47]. Equal-frequency discretization was not used in the test since many datasets have attributes with very low uniqueness, for which the equal-frequency algorithm could not find ten valid bins.

Table 4.7 shows that TUBE-CV is better than the Gaussian estimator in five tests and worse in seven. The simpler EW-CV method wins nine times and loses only five times when compared to the Gaussian estimator. On the datasets where TUBE-CV outperformed the Gaussian estimator EW-CV was superior on four of the five.

In a second set of experiments, an attempt was made to change TUBE's parameters so as to avoid spikes in the histogram, caused by a few instances lying close together. Table 4.8 lists again the accuracy of TUBE cross-validated (TUBE-CV) and TUBE with the cut distance set to 0.1 and the minimal bin width set to 0.2 (TUBE-02). For the results in the third column the maximal

number of bins was set to fifteen (TUBE-15) in addition to the previous settings for the cut distance and minimal bin width. The last column uses a heuristic to avoid spikes in the data and set the maximal number of bins. It builds a cross-validated equal-width histogram first and takes the minimal bin width from the resulting histogram and the maximal number of bins as well (TUBE-EW). The results show that on seven of the datasets the accuracy of the classification improves with all of these variants. But unfortunately performance cannot be improved by these adjustments for most of the datasets.

The third table (Table 4.9) lists the result of further experiments with equal-width varying its parameter settings: the cross-validated equal-width discretization (EW-CV), equal-width discretization with the number of bins fixed to 30 (EW-30) and equal-width discretization with the number of bins fixed to fifteen (EW-15). It can be seen that some results improve, even quite drastically as can be seen in the case of the vowel dataset, but others worsen.

In a fourth table (Table 4.10), naive Bayes with the Gaussian estimator is compared with the best versions from the previous two tables: TUBE with the minimal bin width set to 0.2 (TUBE-02), and equal-width discretization with fifteen bins only (EW-15). TUBE now wins more often: nine times (losing four times) and equal-width discretization wins more often as well: ten times (losing four times). It can be said that the results have improved. This is perhaps because the changes in the distribution are not so sharply represented with these variants, thus the distribution is less overfitted and this improves the predictions.

The last table (Table 4.11) re-states earlier results but in this case the other methods are compared to the best TUBE method, TUBE with the minimal bin width set to 0.2 (TUBE-02). This table shows that only in two cases where TUBE is outperformed by Gaussian the equal-width estimators perform similar to Gaussian (arrhythmia and heart-statlog).

A single Gaussian function is not well suited for fitting multi-modal density functions. An additional experiment investigates the number of modes found in the histograms built by TUBE-15 and EW-15. Table 4.12 lists for those datasets where the binning algorithms performed best (anneal, autos, segment and vehicle), the average number of modes in all non-uniform density estimates (of which there is one for each attribute and each class) for the given datasets.Table 4.13 lists the same numbers for the datasets where the binning algorithms performed worse (heart-statlog, hypothyroid, iris, sick). The histograms with uniform density and therefore zero modes have been left out of

Table 4.7: Naive Bayes with Gaussian density (Gauss) compared with TUBE cross-validated (TUBE-CV), EW cross-validated (EW-CV) and ZeroR (v significant win, * loss against Gauss).

	Gauss	TUBE-CV	EW-CV	ZeroR
anneal	39.50	79.77 v	86.42 v	76.17 v
arrhythmia	61.31	54.20 *	61.97	54.20 *
autos	48.01	73.33 v	77.71 v	32.70 *
balance-scale	90.53	91.44	91.44	45.76 *
w-breast-cancer	96.12	96.77	97.35 v	65.52 *
horse-colic	59.84	65.04	65.11	63.05
german-credit	70.97	67.31 *	66.71	70.00
ecoli	85.50	70.59 *	69.74 *	42.56 *
glass	49.45	56.74	63.24 v	35.51 *
heart-statlog	83.59	63.48 *	72.96 *	55.56 *
hepatitis	80.13.	79.43	78.07	79.38 *
hypothyroid	94.81	93.91 *	95.95 v	92.29 *
ionosphere	82.17	89.57 v	87.80 v	64.10 *
iris	95.53	91.60 *	89.67 *	33.33 *
labor	89.20	84.73	84.03	64.67 *
lymphography	71.91	67.60	69.00	54.76 *
segment	80.13	87.52 v	92.00 v	14.29 *
sick	95.85	91.39 *	96.97 v	93.88 *
sonar	67.71	72.72	65.78	53.38 *
vehicle	44.68	61.41 v	63.04	25.51 *
vowel	67.03	65.97	44.12	9.09 *

Table 4.8: Comparing naive Bayes using TUBE density estimation with various parameter settings for the TUBE estimator: cross-validated (TUBE-CV); cross-validated and 0.2 minimal bin width (TUBE-02); cross-validated, 0.2 minimal bin width and a maximum of 15 bins (TUBE-15); and minimal bin width and maximal number of bins set with EW (TUBE-EW) (v significant win, * loss against TUBE-CV).

	TUBE-CV	TUBE-02	TUBE-15	TUBE-EW
anneal	79.77	79.48	79.49	80.66
arrhythmia	54.20	55.56 v	56.31v	54.51
autos	73.33	75.90	72.01	72.56
balance-scale	91.44	91.46	91.46	91.46
w-breast-cancer	96.77	97.34	97.20	97.25
horse-colic	65.04	65.55	65.28	65.02
german-credit	67.31	67.21	62.45 *	67.89
ecoli	70.59	82.68 v	82.68 v	73.93
glass	56.74	60.24	60.24	64.32 v
heart-statlog	63.48	73.74 v	76.56 v	76.37 v
hepatitis	79.43	79.57	78.45	79.78
hypothyroid	93.91	96.08 v	97.07 v	96.47 v
ionosphere	89.57	90.37	90.37	87.47
iris	91.60	93.60	93.60	90.87
labor	84.73	81.30	81.63	76.10
lymphography	67.60	69.54	69.54	68.87
segment	87.52	91.82 v	91.74 v	92.47 v
sick	91.39	94.94 v	96.29 v	96.72 v
sonar	72.72	75.76	75.76	74.59
vehicle	61.41	62.93	62.50	62.05
vowel	65.97	70.62 v	70.62 v	68.27

Table 4.9: Comparing naive Bayes using equal-width density with various parameter settings for the equal-width estimator: cross-validated (EW-CV); 30 bins (EW-30); and 15 bins (EW-15) (v significant win, * loss against EW-CV).

	EW-CV	EW-30	EW-15
anneal	86.42	86.51	84.78
arrhythmia	61.97	58.63 *	65.22 v
autos	77.71	75.21	68.46 *
balance-scale	91.44	91.44	91.44
w-breast-cancer	97.35	97.37	97.38
horse-colic	65.11	68.84	69.98
german-credit	66.71	68.16	68.83
ecoli	69.74	77.72 v	81.41 v
glass	63.24	64.81	65.07
heart-statlog	72.96	79.63 v	81.30 v
hepatitis	78.07.	76.81	80.48
hypothyroid	95.95	93.70 *	93.44 *
ionosphere	87.80	88.75	88.41
iris	89.67	90.87	94.67 v
labor	84.03	85.40	84.67
lymphography	69.00	69.00	69.00
segment	92.00	92.84	92.45
sick	96.97	97.09	94.79 *
sonar	65.78	69.05	76.77 v
vehicle	63.04	62.13	60.66
vowel	44.12	66.01 v	72.40 v

Table 4.10: Naive Bayes with Gaussian density compared with best of TUBE and best of equal-width estimator: cross-validated and 0.2 minimal bin width (TUBE-02); EW with fifteen bins (EW-15); and ZeroR (v significant win, * loss against Gauss).

	Gauss	TUBE-02	EW-15	ZeroR
anneal	39.50	79.48 v	84.78 v	76.17 v
arrhythmia	61.31	55.56 *	65.22	54.20 *
autos	48.01	75.90 v	68.46 v	32.70 *
balance-scale	90.53	91.46	91.44	45.76 *
w-breast-cancer	96.12	97.34 v	97.38 v	65.52 *
horse-colic	59.84	65.55	69.98 v	63.05
german-credit	70.97	67.21 *	68.83 *	70.00
ecoli	85.50	82.68	81.41 *	42.56 *
glass	49.45	60.24 v	65.07 v	35.51 *
heart-statlog	83.59	73.74 *	81.30	55.56 *
hepatitis	80.13.	79.57	80.48	79.38
hypothyroid	94.81	96.08 v	93.44 *	92.29 *
ionosphere	82.17	90.37 v	88.41 v	64.10 *
iris	95.53	93.60	94.67	33.33 *
labor	89.20	81.30	84.67	64.67 *
lymphography	71.91	69.54	69.00	54.76 *
segment	80.13	91.82 v	92.45 v	14.29 *
sick	95.85	94.94 *	94.97 *	93.88 *
sonar	67.71	75.76 v	76.77 v	53.38 *
vehicle	44.68	62.93 v	60.66 v	25.51 *
vowel	67.03	70.62	72.40 v	9.09 *

Table 4.11: TUBE with the minimal bin width set to 0.2 (TUBE-02) compared with EW using fifteen bins (EW-15), EW cross-validated (EW-CV), and naive Bayes with Gaussian densities(Gauss) (v significant win, * loss against TUBE-EW).

	TUBE-EW	EW-15	EW-CV	Gauss
anneal	79.48	84.78 v	86.42 v	39.50 *
arrhythmia	55.56	65.22 v	61.97 v	61.31 v
autos	75.90	68.46 *	77.71	48.01 *
balance-scale	91.46	91.44	91.44	90.53
w-breast-cancer	97.34	97.38	97.35	96.12 *
horse-colic	65.55	69.98	65.11	59.84
german-credit	67.21	68.83	66.71	70.97 v
ecoli	82.68	81.41	69.74 *	85.50
glass	60.24	65.07	63.24	49.45 *
heart-statlog	73.74	81.30 v	72.96	83.59 v
hepatitis	79.57	80.48	78.07	80.13
hypothyroid	96.08	93.44 *	95.95	94.81 *
ionosphere	90.37	88.41	87.80 *	82.17 *
iris	93.60	94.67	89.67	95.53
labor	81.30	84.67	84.03	89.20 v
lymphography	69.54	69.00	69.00	71.91
segment	91.82	92.45	92.00	80.13 *
sick	94.94	94.97	96.97 v	95.85 *
sonar	75.76	76.77	65.78 *	67.71 *
vehicle	62.93	60.66	63.04	44.68 *
vowel	70.62	72.40	44.12 *	67.03

Figure 4.16: Two lists of instances per bin for the class per attribute sub-datasets of the dataset anneal. The distributions show at least two significant modes.

```
Dataset anneal: 3-rd class, 4-th attribute
Equal Width Estimator. Counts =  8 70 0 0 0 0 0 0 0 11 0 0 0 0 10

Dataset anneal: 3-rd class, 7-th attribute
Equal Width Estimator. Counts =  19 0 0 0 0 0 0 0 0 0 0 0 0 0 80
```

Figure 4.17: Two lists of instances per bin for the class per attribute sub-datasets of the dataset iris. Each distribution shows only one significant mode.

```
Dataset iris, 1-st class. 1-th attribute:
Equal Width Estimator. Counts =  1 3 5 0 7 0 12 8 0 4 0 7 0 0 3

Dataset iris, 1-st class. 2-th attribute:
Equal Width Estimator. Counts =  1 0 0 0 7 5 5 11 6 5 4 2 2 1 1
```

the computation of the average to avoid distorting it. The average number of modes for the first set of datasets is not significantly different from the average for the second set of datasets. Only a closer look at the histograms explains the good performance on the datasets anneal, autos, segment and vehicle. For example, Figure 4.16 lists the instances per bin for two EW fifteen bin histograms from the anneal data. The histograms clearly illustrate the multi-modality of the distributions of the anneal data. In contrast to this, the lists of instances per bin in Figure 4.17 show a distribution that can be represented well by a uni-modal Gaussian distribution. The densities have only one significant mode with some additional small insignificant modes.

4.2.5 Summary Naive Bayes

In this section the univariate TUBE algorithm is applied to naive Bayes classification. The binning of TUBE is used to generate histograms representing the probability distribution of each attribute given a class. A histogram is a nonparametric density estimator that can better represent complex density functions than a parametric density estimator such as the Gaussian distribution. The aim was to improve the accuracy of naive Bayes' predictions by exploiting this property.

Table 4.12: This table repeats the results for the four datasets with the best accuracy for the binning algorithms and adds the average number of modes found: Classification results for Naive Bayes with Gaussian (Gauss), TUBE cross-validated and 0.2 minimal bin width (TUBE-02) and EW with fifteen bins (EW-15); average number of modes found with TUBE cross-validated, 0.2 minimal bin width and maximally fifteen bins (mode-TUBE) and the average number of modes found with EW with fifteen bins (mode-EW).

	Gauss	TUBE-15	EW-15	mode-TUBE	mode-EW
anneal	39.50	79.49 v	84.78 v	3.61	3.27
autos	48.01	72.01 v	68.46 v	3.88	4.24
segment	80.13	91.74 v	92.45 v	1.82	3.63
vehicle	44.68	62.50 v	60.66 v	5.62	4.36

Table 4.13: This table repeats the results for the four datasets with the worst accuracy for the binning algorithms and adds the average number of modes found: Classification results for Naive Bayes with Gaussian (Gauss), TUBE cross-validated and 0.2 minimal bin width (TUBE-02) and EW with fifteen bins (EW-15); average number of modes found with TUBE cross-validated, 0.2 minimally bin width and maximal fifteen bins (mode-TUBE) and the average number of modes found with EW with fifteen bins (mode-EW).

	Gauss	TUBE-15	EW-15	mode-TUBE	mode-EW
heart-statlog	83.59	76.56 *	81.30	3.47	3.50
hypothyroid	94.81	97.07 v	93.44 *	3.10	2.49
iris	95.53	93.60	94.67	1.17	5.16
sick	95.85	96.29	94.97 *	4.17	2.34

It is shown that TUBE can be used for density estimation inside the naive Bayes classifier. It was compared with Gaussian density estimation and two other nonparametric density estimators: equal-width histograms with a fixed number of bins (15 and 30 bins) and equal-width histograms with the number of bins set using cross-validation.

The results show that in some cases the classification accuracy can indeed be improved. However, in most cases the much simpler equal-width estimator performs equally well or even better than TUBE, which is discouraging. With TUBE achieving better results than Gaussian functions on only about half of the datasets, nonparametric density estimation seems to work better only in some cases. Avoiding spikes in the data improves classification accuracy for a number of datasets. This is potentially due to the fact that the density measure used instead of the probability does not correlate at all well with the probability measure at values that fall into areas of spikes.

TUBE histograms are capable of representing the distribution of complex probability functions more closely than the Gaussian distribution. However, this appears to result in an overfitted naive Bayes model in some cases. Hence the aim of improving classification performance could not be reached in general using TUBE as a density estimator for naive Bayes.

4.3 Summary

In this chapter, the univariate TUBE method has been applied to discretization and evaluated firstly as a nonparametric density estimation method. In the first section, about discretization, TUBE histograms were also examined visually to explain how TUBE histograms can represent features like empty areas and abrupt changes in the density distribution. TUBE discretization was compared with other unsupervised discretization methods, namely equal-width and equal-frequency discretization, using the log-likelihood criterion. TUBE gave similar results to these methods and better results for very discontinuous data.

However, a caveat needs to be attached to these findings: experiments with different values of the minimal bin width show that on some of the test datasets a small value can strongly influence the log-likelihood of the TUBE histogram.

The second evaluation of TUBE discretization was as a density estimation tool inside the naive Bayes classifier. Naive Bayes predicts the class value using probability measures estimated from the data. A standard method is to use a Gaussian density function. TUBE histograms, like other nonparametric density estimators, can infer more complex probability structures. However, the comparison of naive Bayes prediction accuracy using TUBE histograms compared with naive Bayes using Gaussian distribution functions and other nonparametric density estimators based on equal-width discretization could not fulfil the expectation of improving naive Bayes classification in general. Overfitting of the density function may be the reason that TUBE discretization is not a general-purpose tool for density estimation inside naive Bayes

The next chapter, Chapter 5, introduces the multidimensional TUBE algorithm, Multi-TUBE. Multi-TUBE can be applied to clustering and mode finding for density estimation. Applications of Multi-TUBE are investigated in Chapter 6.

Chapter 5

Multivariate Density Estimation Trees

This chapter discusses a variant of the basic TUBE tree building algorithm that builds a density estimation model for multivariate data. The basic TUBE algorithm for building a tree model for density estimation from univariate data is introduced in Chapter 3. The algorithm for multivariate data is called *Multi-TUBE*.

Multi-TUBE stands for Multidimensional Tree-based Unsupervised Bin Estimator. Multi-TUBE, like TUBE, creates a binary density estimation tree by splitting the dataset recursively. It is also an unsupervised method and thus does not take class values into account when building the tree model. Each inner node of the tree represents a split along one value of one of the attributes in the data. In this way the algorithm cuts the total range into axis-parallel multidimensional bins, which represent areas of approximately uniform density. The combined bins form a step-wise constant density function blanket and can be viewed as a multidimensional histogram. Like the one-dimensional TUBE histogram, this multidimensional histogram can be used for density estimation.

In the field of multidimensional density estimation the *curse of dimensionality* is an important issue, as discussed in Section 2.1.2. It is easy to see that, because of the curse of dimensionality, an adaptation of the equal-width histogram to the multidimensional case is not a feasible approach for many cases. To see why, consider the data model with four numeric attributes from Section 2.1.2. A general heuristic for equal-width histograms is to choose the number of bins between 10 and 30. Let us assume that each attribute is divided into ten equal-length subranges. Let n be the number of attributes and

k be the number of subranges. The number of resulting multidimensional bins is k^n and for this example it is $10^4 = 1000$. The resulting multidimensional histogram for this problem has 1000 bins. Consequently the datasets must contain substantially more than 1000 instances; otherwise only a few instances would be in each bin and the density estimate would be distorted.

In datasets with a large number of attributes it is likely that some attributes do not contribute to the overall density in a meaningful way because they are uniformly distributed. Also, groups of attributes can be strongly correlated. Thus the question arises as to how to ignore these irrelevant attributes. Several attribute selection methods have been developed for preprocessing a dataset before a classification, clustering or other machine learning task is performed. However, there is little work on unsupervised attribute selection. TUBE generates a density model by searching for the most significant changes in density. Multi-TUBE does the same as TUBE, but across multiple attributes. When selecting the next best split, it searches for the most significant change in density, for each of the attributes in the data, and consequently automatically chooses between attributes when splitting. Hence Multi-TUBE can avoid irrelevant attributes effectively by not splitting on them.

The following sections of this chapter explain the Multi-TUBE method in more detail. They largely cover the same topics as in the univariate case, while highlighting the required adaptations to multidimensionality. More specifically, this chapter discusses the evaluation methods used for the binning steps (Section 5.1), the splitting process (Section 5.2), and the construction of the density estimation tree (Section 5.3)—including attribute selection 5.3.1, stopping criteria (Section 5.3.2) and the problem of narrow cuts (section 5.3.3). Section 5.4 introduces a feature that is specific to Multi-TUBE: the mixing of binnings. Section 5.4 also briefly mentions using Multi-TUBE for clustering, but TUBE-based clustering will be explained in more detail in the next chapter. Section 5.5 discusses new representation techniques developed for Multi-TUBE's multidimensional histograms and uses these techniques to explore a few datasets.

5.1 Evaluating a Multidimensional Binning

Like TUBE, Multi-TUBE uses the log-likelihood criterion to evaluate a single split and the whole binning corresponding to a particular tree. Since Multi-TUBE's bins do not only have width but volume, the computation has to be

generalized.

Multi-TUBE cuts the space into multidimensional bins, and the density of these bins forms a multidimensional blanket and thus represents the estimated density function. In TUBE, the density (i.e. height) d_i of bin_i is computed using the bin width w_i and the number of training instances n_i falling into bin_i:

$$d_i = \frac{n_i}{w_i \times N}$$

In Multi-TUBE the width of the bin is substituted by its volume v_i. With this, the density d_i of bin_i is:

$$d_i = \frac{n_i}{v_i \times N}$$

To avoid very large values for the volume each attribute range is implicitly normalized to length 1.0. More specifically, the volume v_i is computed using the width $w_{i,j}$ of bin i for attribute i and max_j, the length of the full range for attribute a_j. Then the volume of bin_i is:

$$v_i = \prod_j \frac{w_{i,j}}{max_j}$$

5.2 Splitting a Range and Setting the Cut Point

Multi-TUBE uses the same principles as TUBE for splitting a range, but considers all attribute ranges of a bin for each split decision. To achieve this, each attribute must be sorted independent of the other attributes. Each attribute is only sorted once and further split operations adapt the begin and end index in the sorted array of values according to the bin they are working in.

For the first split, which forms the root node, univariate TUBE searches the full range of the attribute in the data, compares all possible split points using the log-likelihood criterion and selects the best point as the next possible split. Multi-TUBE does exactly the same, but for each of the attributes individually. It then uses the criterion of log-likelihood improvement (i.e. likelihood gain) to select between the options. More specifically, assuming a training dataset with k attributes, it decides between k attributes for the first split in the root node. Note that, after this split has been performed, the other $k-1$ computed cut points become invalid since the instances they have been computed with have

been separated into two different bins. This process is repeated recursively as described in the next section.

Where does Multi-TUBE cut when considering a certain attribute? The cut is performed in exactly the same way as in the univariate case: at a fixed position before or after a particular instance (cuts are axis-parallel). In a dataset with N instances, Multi-TUBE has $2 \times (N - 1)$ cut points to test (and one more cut point at each end of the range if the empty space between the most extreme instances and the corresponding end point is larger than the fixed distance Multi-TUBE is cutting with). In the implementation of the algorithm used for the experiments presented later, the fixed distance was set to 10^{-5} as in TUBE. The value can be set with a user parameter.

5.3 Building the Density Estimation Tree for Multidimensional Data

The multidimensional Multi-TUBE algorithm builds a density estimation tree in a very similar way as one-dimensional TUBE. Multi-TUBE considers each of the attributes for each split. However, the splits are done in an axis-parallel fashion which means only one attribute is involved in each split. In the following, the tree building algorithm is first explained in detail. A small simplification is given at the end of the section and the pseudo code in Algorithm 2 contains the final algorithm with this simplification.

As a first step, Multi-TUBE searches the full range for the best split, as explained above. In contrast to TUBE, Multi-TUBE does this not just once but, in a dataset with k attributes, k times. All k computed splits are added to the priority queue. Then the best split from the top of the queue is taken and performed. The information corresponding to this split is stored in the root node of the tree. As in TUBE, the priority queue holds all prepared splits ordered by the improvement in log-likelihood that they yield.

Unlike TUBE, Multi-TUBE has to delete some of the computations from the queue, because the split of the dataset renders them void. For the first split, the $k - 1$ computations for the other attributes have to be discarded even though the split was not performed in their respective dimensions: the instances are now partitioned into two bins, which makes the computations invalid. This can be shown using an example. Consider a two-dimensional dataset with attributes $a1$ and $a2$. In Figure 5.1, no split has been performed

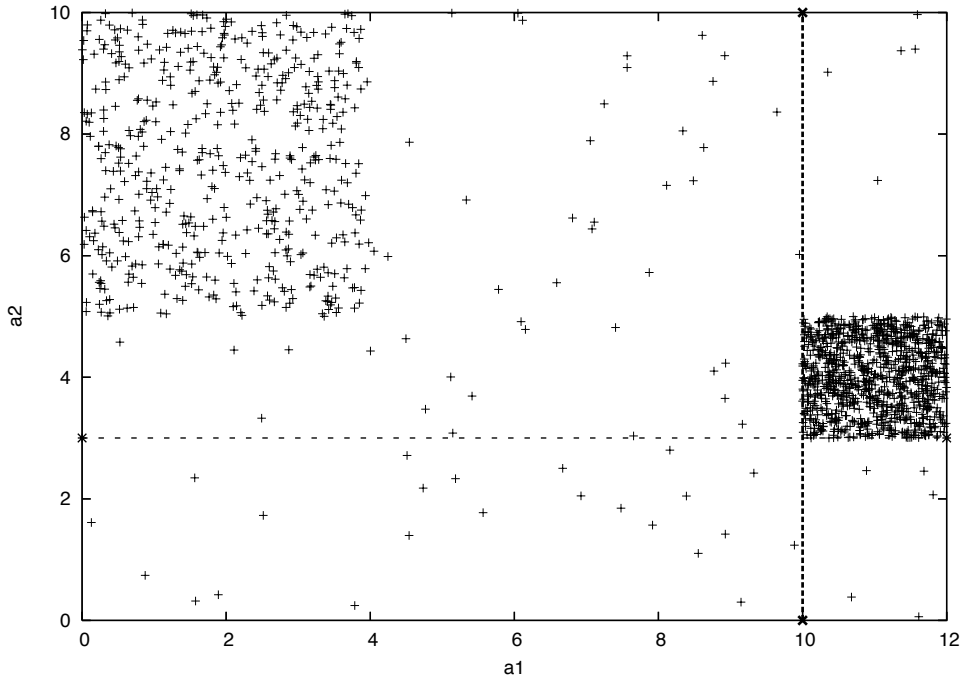

Figure 5.1: Two-dimensional dataset with possible cut points before first cut.

yet, and the two best computed splits for the two dimensions are drawn as dotted lines. The split on $a1$ turns out to be the better one according to the likelihood improvement measure. Figure 5.2 shows the two best splits for the left bin after $a1$ has been split at value 10. Note that the new split for attribute $a2$ is in a different position than the discarded split on $a2$, which was computed for the full range. This shows that the prepared splits have to be discarded since the distribution in one part of a bin is never completely identical to the distribution in the whole bin. This applies to all splits, not only the first split.

Consequently, after cutting a multidimensional range, Multi-TUBE deletes the $k-1$ unused splits of the bin. Then it searches the two resulting bins for the best possible splits in each of their k attribute ranges and adds all $2 \times k$ new splitting canditates to the priority queue. This process is repeated recursively. This way Multi-TUBE builds a binary density estimation tree similar to TUBE's density estimation tree. Each inner node of the tree represents one split and stores not only the split value but also the attribute it splits on. The leaf nodes are the resulting bins.

The way Multi-TUBE selects the next node to expand by comparing all possible splits using a criterion is again a greedy way of finding a locally optimal solution. Multi-TUBE, like TUBE, does not perform backtracking, but relies

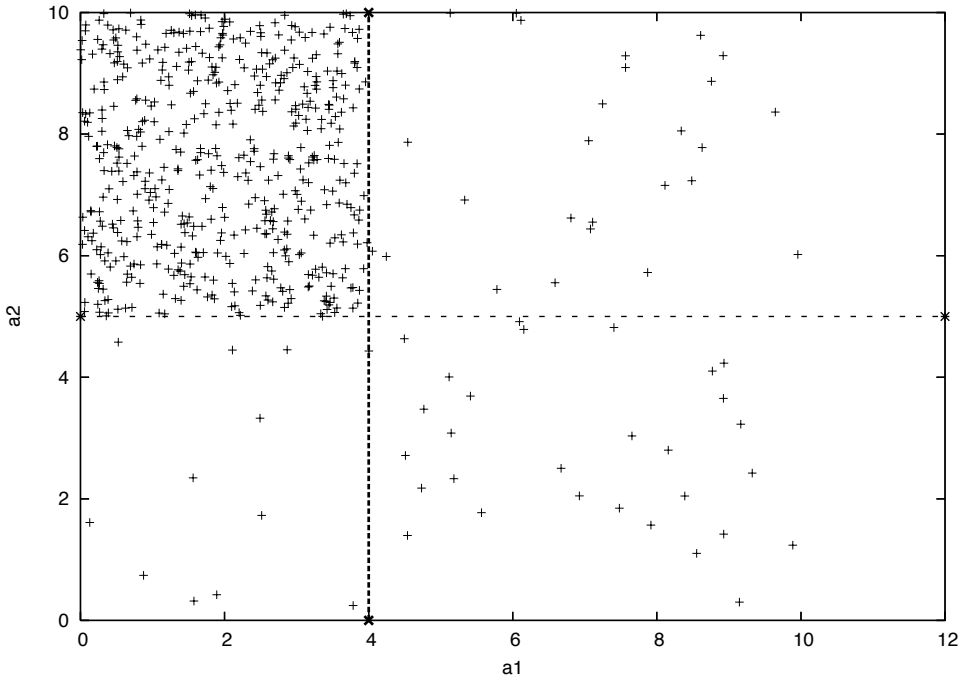

Figure 5.2: Left bin after first cut, with two new cut points for $a1$ and $a2$.

on a sufficiently good, non-optimal result.

As mentioned at the beginning of this section, the process of adding computed splits to the priority queue can be simplified. In the process described so far, k split candidates are added to the priority queue for each successor bin. But, because only the best of the k splits is performed and all other splits have to be discarded once this split has been executed, it is more efficient to simply select the best split and add only that split to the priority queue. This makes it unnecessary to delete the other split candidates after the split has been done. This simplified algorithm is implemented in Multi-TUBE and shown as the pseudo code in Algorithm 2.

5.3.1 Selecting the Next Attribute to Cut

In Multi-TUBE, the algorithm implicitly decides which attribute to cut by comparing the k possible split candidates from each of the current leaf nodes and then selecting the best of them. Similar processes have been used for decision tree construction [50]. Multi-TUBE uses a priority queue-based best-first expansion strategy to place an order on the nodes so as to enable cross-validation-based pruning.

At each step, Multi-TUBE selects the attribute that defines the most sig-

Algorithm 2 Pseudo code for the basic Multi-TUBE binning algorithm.

```
maxNumBins ⇐ CV-ForOptimalNumberOfBins();

numSplits ⇐ 0; {Counts current number of splits}
bestLocalSplit ⇐ new Split; {Auxiliary variable to store best local split}
bestLocalSplit.LLKGain ⇐ 0.0; {Initialize log-likelihood-gain}
splitPriorityQueue ⇐ empty; {Priority queue stores next possible splits}

firstBin ⇐ new Bin; {Bin which contains the whole attribute range}
binList.add(firstBin); {List to gather all bins}
for i over all k attributes do
  split ⇐ bin.findBestSplitInAttribute(i); {Best split for i }
  if split.LLKGain > bestLocalSplit.LLKGain then
    bestLocalSplit ⇐ split
  end if
end for
splitPriorityQueue.add(bestLocalSplit);

while numSplits + 1 < maxNumBins do
  nextBestSplit ⇐ splitPriorityQueue.top(); {Best split in queue}

  {* Perform the split on the bin *}
  {* and replace the bin in bin list with two new bins *}
  newBinLeft, newBinRight ⇐ nextBestSplit.performSplit(binList);
  numSplits + +; {One more split done}

  {* Finds the best possible split in the range of the new left bin ..*}
  for i over all k attributes do
    split ⇐ newBinLeft.findBestSplitInAttribute(i); {Best split for i }
    if split.LLKGain > bestLocalSplit.LLKGain then
      bestLocalSplit ⇐ split;
    end if
  end for
  splitPriorityQueue.add(bestLocalSplit); {Adds it to the priority queue}


  {* .. and in the range of the new right bin *}
  for i over all k attributes do
    split ⇐ newBinRight.findBestSplitInAttribute(i); {Best split for i }
    if split.LLKGain > bestLocalSplit.LLKGain then
      bestLocalSplit ⇐ split;
    end if
  end for
  splitPriorityQueue.add(bestLocalSplit); {Adds it to the priority queue}
end while
return binList
```

nificant density changes in the dataset to find areas of equal density. This way it automatically ignores uniformly distributed attributes and attributes that are correlated to other attributes. In practice, it sometimes only cuts a small fraction of the attributes of a dataset to create its binning.

5.3.2 The Stopping Criteria

Multi-TUBE uses the same stopping criteria as TUBE, the cross-validated likelihood and the local penalty rule discussed in Section 3.6. For the local penalty rule to apply to a node in the case of Multi-TUBE, all attributes' best splits must be rejected so that the node is not expanded further. The number of node expansions that are performed can be selected in the same fashion as in TUBE, using cross-validation, because the same best-first node expansion process is applied.

5.3.3 The Problem of Narrow Cuts

The tendency of Multi-TUBE to produce narrow bins was investigated using nine real-world datasets. (These datasets are again used in the multiple-instance application of Multi-TUBE described in the next chapter. Details on these datasets can be found in Section 6.2.5. For the purposes of this experiment they were transformed from multiple-instance format to standard data format).

All datasets were split using Multi-TUBE, based on a fixed number of bins. The resulting histograms were inspected manually. Interestingly, the results showed that there were no narrow cuts that resulted in a distortion of the histograms. Although there were no extreme distortions due to narrow bins, the histograms were generally quite skewed, with one mode bin that contained many instances in one small bin—between 10 and 90 percent of the instances, in a bin with a volume smaller than 1 percent of the total volume.

5.4 Additional Functionality in Multi-TUBE

In this thesis, Multi-TUBE is used in two applications, which are both investigated in detail in Chapter 6. For these applications, the combination of cuts from two histograms and a clustering algorithm respectively were implemented. The combination algorithm is explained in detail in the following

subsection (Section 5.4.1). The clustering algorithm is rather complex and constitutes an application in itself and is discussed in detail in the next chapter, but Section 5.4.2 gives a brief preview.

5.4.1 Mixing Two Binnings

In tasks like classification of data with a binary class it can be important to find areas in the distribution of the data where the density of one class value is high, not absolute but in relation to the density of the second class value. Two densities over the same range can easily be compared when represented by two density histograms, if it is ensured that each bin has the same position and size as the corresponding bin of the other histogram. For a comparison of two densities, a 'difference' function can be used; for instance, the following function $D(x)$, which defines the difference between the two densities $f(x)$ and $g(x)$:

$$D(x) = f(x) - g(x)$$

The function D of any two histograms can be generated by setting the height of each bin to $d_{i,f} - d_{i,g}$, with $d_{i,f}$ being the height of bin_i of histogram f and $d_{i,g}$ being the height of bin_i of histogram g. In up to two dimensions, the resulting difference function can be visualized. Note that the ranges of the two histograms might have to be consolidated in the resulting histogram and resulting bins can have negative heights: the difference function D of two histograms is not a density histogram itself.

The process necessary for computing this kind of difference function is called *mixing of two binnings*. It was implemented to be able to compare two histograms (one-dimensional and multidimensional ones), which do not have identical cuts. The mixing of binnings requires a process of 'unifying' two binnings so that there is a direct correspondence between the resulting bins. The mixing process does this by simply adding the cuts of the second binning to those of the first.

This section first gives an example, illustrating the outcome of combining two one-dimensional binnings using the mixing algorithm. It then explains the implementation of the multidimensional mixing process. Later it discusses an upper bound on the number of bins that mixing can yield and under which conditions this maximum is reached. Finally, this section discusses a technical

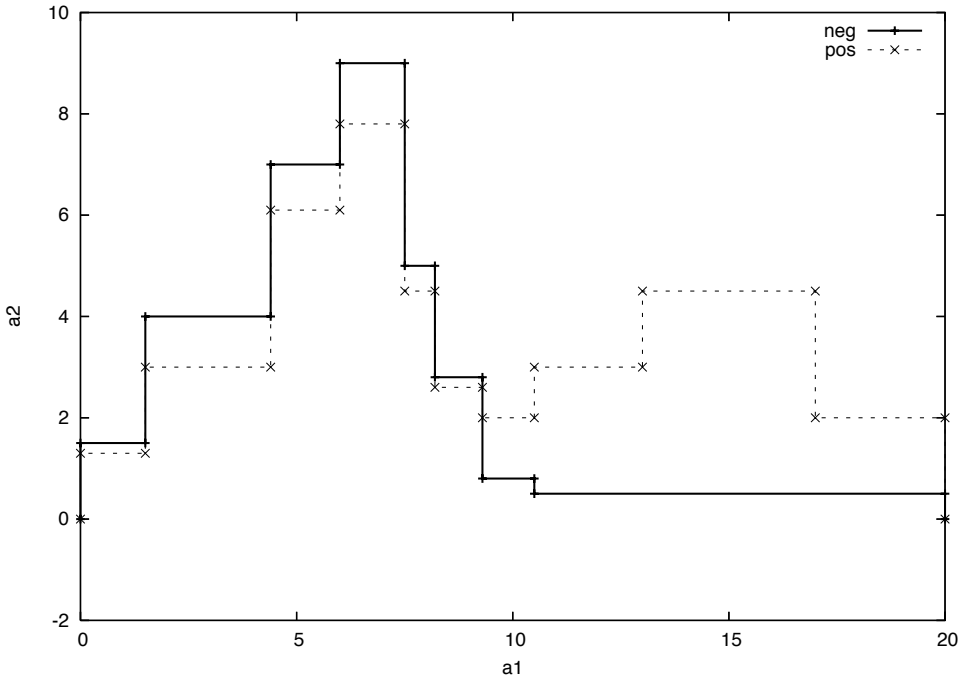

Figure 5.3: Two univariate distributions: dataset with 'negative' instances and dataset with 'positive' instances.

problem concerning the mixing process. The feature 'mixing of two binnings' is used in the application of Multi-TUBE to multiple instance learning, which is discussed in detail in the next chapter (Chapter 6).

Example: Mixing Two One-dimensional Binnings

In this example, two different datasets distributed over the same range [0.0..10.00] have been discretized using one-dimensional TUBE, which resulted in two independent binnings with bins of varying length. The histograms are both shown in Figure 5.3. The function D, the corresponding difference function, is given in Figure 5.4. It can be seen that it has some bins of negative 'height' .

Implementation of the Multidimensional Mixing of Binnings

Compared to the mixing performed on one-dimensional datasets, mixing multidimensional binnings is more complicated. In the one-dimensional case, any additional cut only affects one existing bin. When two multidimensional binnings are mixed, each cut can disect several bins. A simple two-dimensional example illustrates this.

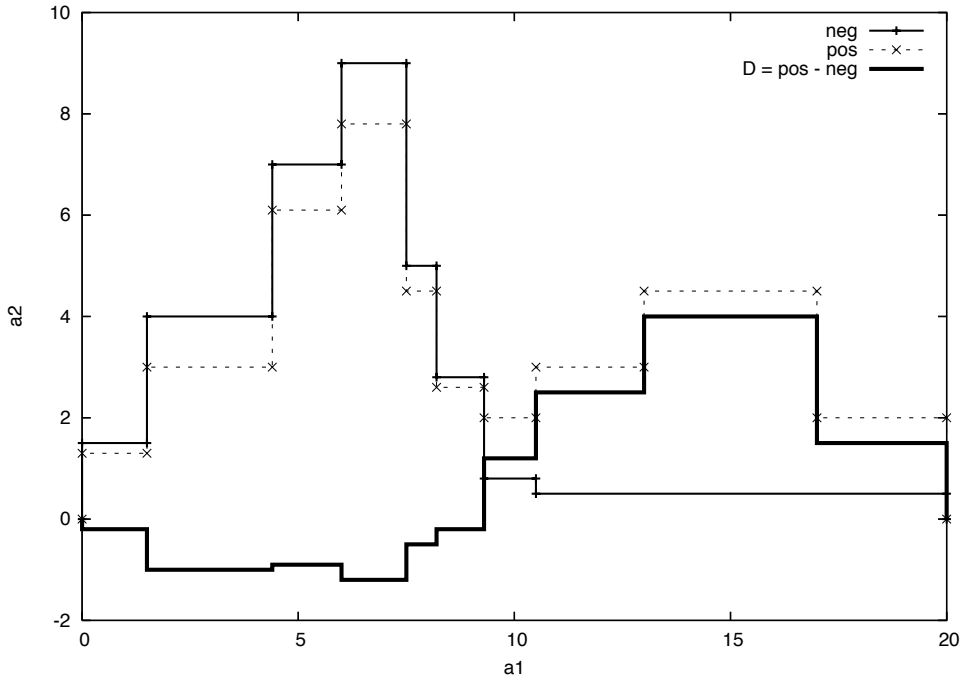

Figure 5.4: Two univariate distributions and their difference distribution D_{diff}.

Example: Mixing Two Two-dimensional Binnings Consider two datasets that are spread over the same two-dimensional range. One dataset has instances of class A, the second dataset has instances of class B. For each dataset, Multi-TUBE was used to generate a binning (called binning A and binning B in the following). Figure 5.5 shows how the range was split for dataset A and Figure 5.6 how it was split for dataset B. If the cut along attribute $X1$ of binning B is performed on dataset A, two of the bins in binning A are split. Taking the next cut in binning B and applying it to binning A shows one bin from binning A is cut fully and the other relevant bin has already been cut with the first cut from binning B and therefore only a part of it is cut. Figure 5.7 shows the result of mixing the two binnings.

Tree Structure of the Combined Binning As explained above, two binnings built by Multi-TUBE on datasets with the same set of attributes can be combined into one binning. However, the current implementation of the mixing algorithm does not actually construct a tree structure that is a model for the new binning. If such a tree structure were needed a simple way to build it would be to attach the tree of the second binning to each of the leaf nodes of the tree corresponding to the first one. After the leaf nodes have been replaced, all these newly attached trees can be pruned, removing redundant

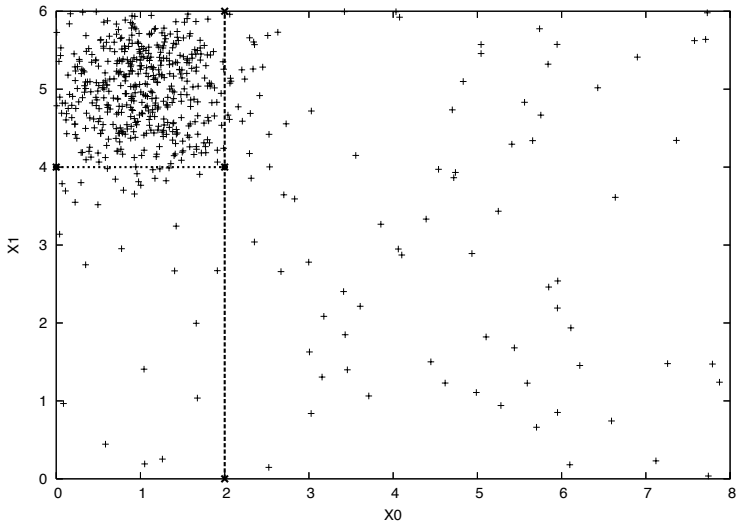

Figure 5.5: Two-dimensional dataset with class A instances and two splits.

nodes. A redundant node is for example a node with a condition $a_1 < 7.0$ when in the path to this leaf node a condition $a_1 < 5.0$ already exists.

Overview of the Implementation of the Mixing Algorithm To combine two multidimensional binnings Multi-TUBE's mixing of binnings feature first defines a total range by taking the maximum of the ranges for each attribute. It then takes the bins of the first binning and cuts them using the cuts defined by the second binning, starting from the top of the tree. For this it uses both tree structures but does not build a new tree structure. This process results in a new set of bins with some or all of the bins of the first binning being refined into smaller bins. See also Algorithm 3

The Maximal Number of Bins after Mixing

This section examines, what is the maximum number of bins that can result from mixing, to check whether there is a danger of shredding the multidimensional space into too many small bins when mixing two binnings.

Assume there are two multidimensional density estimation trees, N and M, generated by Multi-TUBE, which represent a multidimensional binning: Tree M with m cuts and tree N with n cuts. Each cut splits one bin into two bins and hence adds one more bin. Therefore the binning represented by tree M has $m+1$ multidimensional bins and that of tree N has $n+1$ multidimensional bins. What is the maximal number of bins that can result when combining the cuts of both trees?

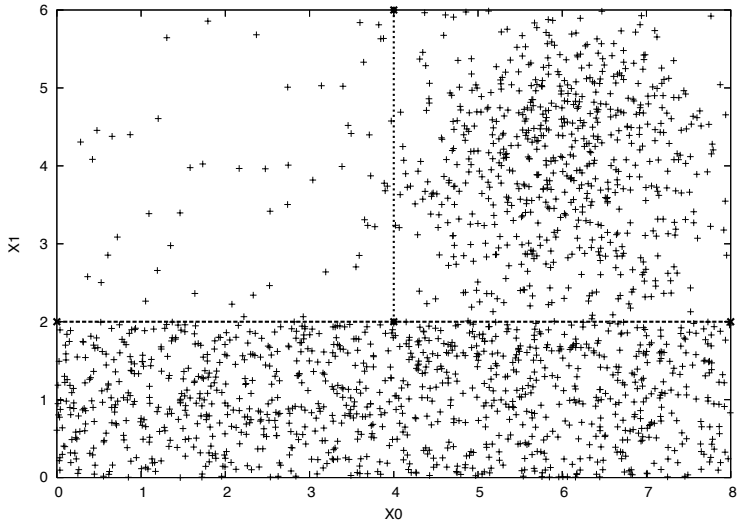

Figure 5.6: Two-dimensional dataset with class B instances and two splits.

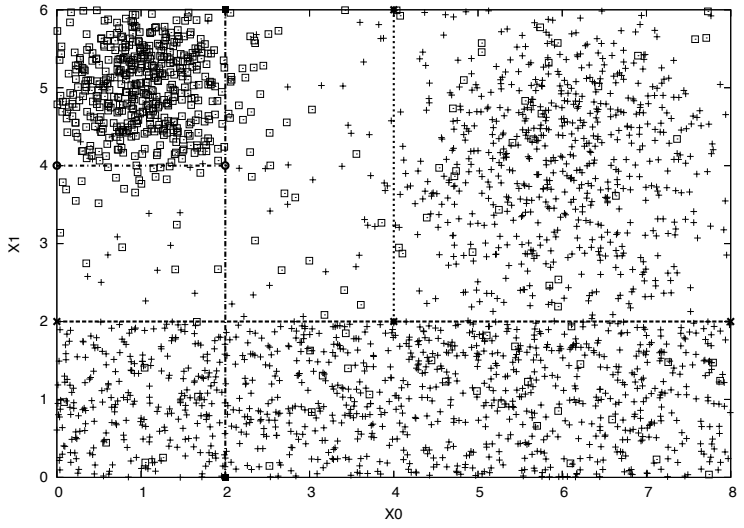

Figure 5.7: Dataset with class A and class B instances and mixing of binnings performed.

Algorithm 3 Pseudocode: The mixing of binnings algorithm combines two binnings (A and B) by recursively applying the cuts stored in cut tree B to all bins in bin list A (current implementation does not change cut tree A)

function OneCut(*node*, *bins*; list of all leave bins to be cut)

for bin_i in *bins* **do**
  if bin_i is within subrange of *node.innerBin* **then**
    if *node.cutValue* on attribute *node.cutAttr* is within subrange of bin_i **then**
      perform cut and replace bin_i in *binListA* with the two new bins
    end if
  end if
end for

perform function OneCut(*node.nodeLeft*, *binListA*)
perform function OneCut(*node.nodeRight*, *binListA*)
end of function

main()
binningA: *binListA* (all leave bins) based on *cutTreeA*
cutTreeB
node: *cutAttr*, *cutValue*, *innerBin*, *nodeLeft*, *nodeLeft*

actualNode = *rootNode* of cut tree B
perform function OneCut(*actualNode*, *binListA*)
end of main

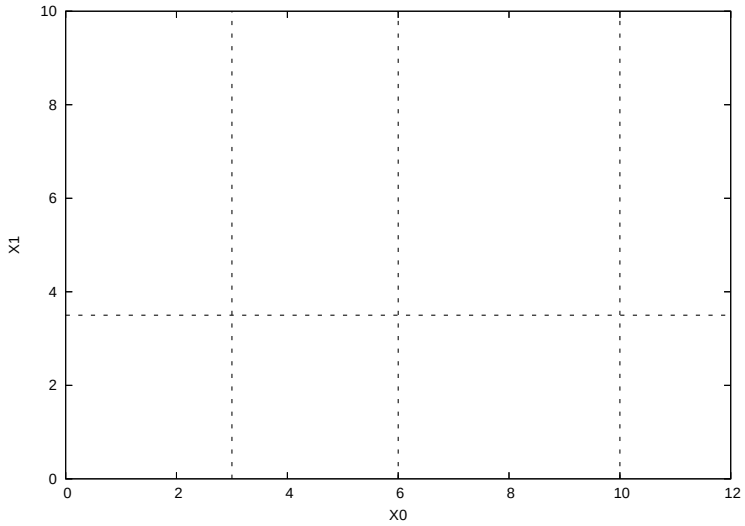

Figure 5.8: The first mixed cut is cutting an attribute ($X1$) that has not been cut yet and therefore cuts all existing bins.

A new cut that uses an attribute that has never been used before, always splits all existing bins (see Figure 5.8). Therefore, it results in the maximum increase in the number of bins. Based on this fact and in each step considering the worst case the following will show which conditions must hold so that a mixing of two binnings M and N results in the maximum number of bins.

First Cut: In the worst case, the first cut from the root node of tree N is on an attribute that is not used in any cuts in M and therefore cuts all bins and so adds $m + 1$ new bins. The resulting binning has now $2 \times (m + 1)$ bins.

Second Cut: The next cut taken from tree N (from either of the two branches extending from the root node) and applied to tree M's binning is in another new dimension not used in M and not used in the first cut in N. It cuts $m + 1$ bins and adds another $m + 1$ bins, which is the maximum number of bins a second cut can add.

Continuing to Add Cuts from N to M Multi-TUBE continues splitting the bins recursively, and if all further cuts in N are each in a further new dimension, they each add $m + 1$ bins.

Result: In the worst case, each of the n cuts of tree N adds $m + 1$ bins to M. Therefore the result is that the maximum number of bins after mixing the binnings of trees M and N is:

$$max = (n + 1) \times (m + 1)$$

Conditions for Maximum Number of Bins The necessary conditions under which the maximum number of bins in the resulting binning is obtained, is that all cuts in N are on different attributes and on attributes different from those used in M. This can only be the case when the number of attributes k in the dataset is larger or equal to the number of cutting attributes in N plus the number of cutting attributes in M.

Example with m = 9 and n = 9 If $m = 9$ and and $n = 9$, which would mean that both trees M and N represent a binning with 10 bins, the maximum number of bins after mixing the two binnings is $10 \times 10 = 100$. This maximum number is reached if each cut in N uses a different attribute, all of the attributes used in N are different to the attributes used in M, and the number of attributes is 10 or greater.

Conclusion The above shows that the multidimensional mixing of binnings can, under certain conditions, shred the range into many more bins than the original two binnings had.

Problem: Identical Cut Points with Different Bounds

The bounds of intervals can either be defined as 'open', which means that the value at the bound is not part of the interval, or 'closed', when the value at the bound is part of the interval. Open bounds are denoted by '(' or ')' and closed bounds by '[' or ']'. Because histogram bins are mutually exclusive intervals, the interval bounds of two adjoining histograms are always different—e.g. ..), [..—and thus leave a value that is located exactly at the boundary of the bins in either the left or the right bin.

When mixing the binnings of two histograms, the problem can arise that both histograms have a cut point in common but the respective bounds are defined differently in each of the histograms. For example, two one-dimensional two-bin histograms could have their bins defined as:

```
Histogram-1: [2.0 .. 5.0), [5.0 .. 7.0]
Histogram-2: [2.0 .. 5.0], (5.0 .. 7.0]
```

A possible solution is to define a new bin around the problematic value using the fixed cut distance employed in TUBE's cutting process to form a new bin.

In practise this problem very rarely arises because TUBE does not cut at the value of an instance, but at a fixed distance from the training instance, when building the density estimation tree. If two cut points are nevertheless identical, the first setting of bounds is taken. A test instance could be exactly on the cut value, but arbitrarily the impact of this inexactness was considered marginal.

5.4.2 Clustering Using the Multi-TUBE Binning

Clustering data means to identify high density areas surrounded by low density ones. The high density areas correspond to modes (peaks) in the density. Multi-TUBE's multidimensional histogram can be used to search for clusters in multidimensional datasets. The clustering algorithm developed in this thesis uses Multi-TUBE's binning to find modes in the density. It searches for 'mode' bins and gathers all neighbouring bins down to the 'valley' bins to form clusters. The TUBE-based clusterer is a density-based clusterer. It is explained in detail in the next chapter (Chapter 6).

In the remainder of this section tools for inspecting the output of Multi-TUBE are discussed.

5.5 Presentation Methods for Multidimensional Bins

For centuries the histogram has been a very popular tool to represent data. However, multidimensional data is difficult to represent in a visualization. For the analysis of the results in this thesis it was desireable to have some visual means of gaining insight into the structure of the data. Visualization can aid in finding explanations for results of a machine learning step like classification.

The matrix plot is a common method to represent multidimensional data. It consists of scatter plots considering all combinations of the attributes to form a matrix of two-dimensional scatter plots. But, for datasets with many attributes, perhaps more than 100, the matrix plot gets too large to make an overview possible. There is also the problem of having many instances in the datasets. In a scatterplot, each instance is plotted as a dot. When plotting

the instances as black dots on a white background, the user can identify local densities as the variation in the darkness of the area. The darkest areas are the densest. But if the number of instances is very high, some areas can be blackened out, and no information about density variation is gained from the plot. The histogram obviates this problem with large datasets because it is a smoothed version of the scatterplot.

The multidimensional TUBE algorithm can be used to construct histograms. To represent these histograms, two simple semi-graphical representations—the *bin list* and the *bin position overview*—have been developed. Both methods are not restricted by datasets with many attributes and can also be used with very large datasets. The two diagram techniques are explained in the following subsections, and are used in the chapter on "Applications of Multivariate Density Estimation Trees" (Chapter 6) to document the results obtained in the Multi-TUBE applications.

5.5.1 Ordering the Bins

In contrast to the one-dimensional binning, there is no order on the bins in the multidimensional case. In the new representation technique bin list, explained in the following, the bins are ordered based on the tree generated by Multi-TUBE. The left-most bin in the tree is the bin for which all subranges start from the minimum of the range of the corresponding attributes. This is because, if a range is cut, the part with the smaller values is located on the 'left' side of the cut (traditionally the real line is represented from the smallest values on the left to the larger values on the right). With this order given, a histogram could be drawn and the width of each bin set relative to the volume of this bin. Instead, the semi-graphical representation technique called bin list was developed, which can give even more information to the user.

5.5.2 The Bin List: A Simplified Histogram

Multi-TUBE splits the range into areas, choosing the sizes of the bins in such a way that they adapt to the local density. The resulting bins show the most significant features of the distribution of the instances. Instead of drawing the multidimensional bins with the width relative to the volume, the representation technique *bin list* was developed. It uses a textual representation.

More specifically, one line of text in a bin list gives the most important information for one bin. These are: density, volume and the percentage of

```
Bin List - with instances percent:
Highest Mode Bin: 3

0  : Dns:[          ] Ins:[          ] Vol:[X.........]
1  : Dns:[X.........] Ins:[X.........] Vol:[XX........]
2  : Dns:[XXXX......] Ins:[X.........] Vol:[X.........]
3  : Dns:[XXXXXXXXXX] Ins:[XXXX......] Vol:[X.........]
4  : Dns:[X.........] Ins:[X.........] Vol:[X.........]
5  : Dns:[X.........] Ins:[X.........] Vol:[X.........]
6  : Dns:[XXXX......] Ins:[XXXXX.....] Vol:[X.........]
7  : Dns:[X.........] Ins:[X.........] Vol:[X.........]
8  : Dns:[X.........] Ins:[X.........] Vol:[X.........]
9  : Dns:[X.........] Ins:[X.........] Vol:[XXXXXX....]
Percentage of instances presented(Dns): 100%
```

Figure 5.9: A sample bin list of ten bins; Bin 0 is empty.

```
...
15 : Dns:[<1E-3.....] Ins:[<0.01.....] Vol:[XXXXXXX...]
Percentage of instances presented(Dns): 99.87%
```

Figure 5.10: Values below 0.1%.

instances that the bin contains. Each of these values is not given as a number but as a list of `X`-characters (exceptions to this rule will be explained below) aligned in a way that the density values, for example, of all bins form a column. This way columns of these values can be seen as small vertical histograms. Next to the density information, two more lists are given that show the relative volume (=‘width’) of the bins and the relative number of instances in each bin respectively. An example bin list is given in Figure 5.9.

In a multidimensional histogram, as mentioned above, the order of the bins is not clearly defined, as it is for the one-dimensional histogram. In the bin list, the order of the bins is determined by gathering the leaves of the tree generated by Multi-TUBE from left to right: smaller value subranges to larger value subranges.

For the examples considered below, a dataset was generated with instances that have six attributes. The distribution in this dataset consists of a few areas of uniform density. The matrix plot in Figure 5.11 shows this dataset as a matrix of scatter plots.

Using the Multi-TUBE binning, the generated dataset was split into nine bins and the resulting multidimensional histogram is represented in the bin

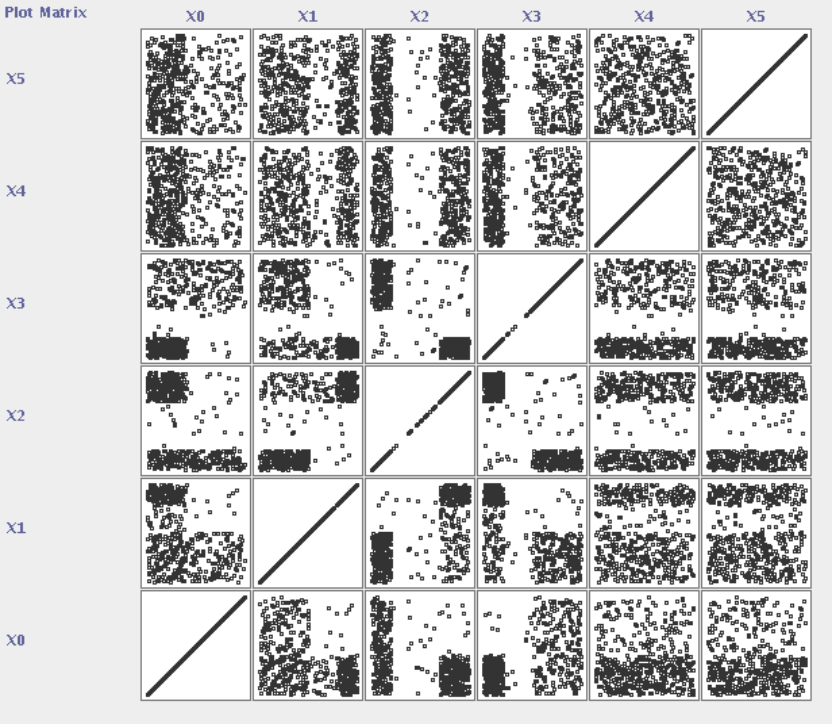

Figure 5.11: Matrix plot of the example dataset.

list in Figure 5.9. As can be seen, the representation of the bins is strongly simplified. Each bin is given a number for further reference. One line in the bin list shows the following values for the corresponding bin: density (Dns), percentage of number of instances in the bin (Ins), and percentage of volume (Vol), forming three columns. All three values are represented in a semi-graphical way using `X` characters. One `X` stands for 10 percent. The volume is given as the percentage of the total volume, and the number of instances as the percentage of all instances. The percent values are rounded up to the next 10. So if the percentage is 53.0 percent, six `X` characters are drawn. Consequently the number of `X` characters seen across all rows do not add up to ten.

For the density values nothing like a total sum exists. For the presentation of the density, the density of the 'highest' bin—the bin with largest density across all bins—is taken as 100 percent and the densities of the other bins are given as a percentage compared relative to the density of this bin. Therefore the densest bin is represented with `[XXXXXXXXXX]`, a string containing ten `X` characters.

All values are rounded up to the next 10, but values that are smaller than `0.1` are not shown as `[X.........]`, instead they are written as `<0.1` or `<0.01` continuing down to `<1E-28`. (See two examples in Figure 5.10.) In practise,

```
0  : -.........- Dns:[          ] Ins:[          ] Vol:[X.........]
1  : Ab.........B Dns:[X.........] Ins:[X.........] Vol:[XX.......]
2  : BaaaaaaaaaaA Dns:[XXXX......] Ins:[X.........] Vol:[X.........]
3  : ObbbbbbaaaaA Dns:[XXXXXXXXXX] Ins:[XXXX......] Vol:[X.........]
4  : Ab.........B Dns:[X.........] Ins:[X.........] Vol:[X.........]
5  : Ab.........B Dns:[X.........] Ins:[X.........] Vol:[X.........]
6  : Abbb.......B Dns:[XXXX......] Ins:[XXXXX.....] Vol:[X.........]
7  : Ab.........B Dns:[X.........] Ins:[X.........] Vol:[X.........]
8  : Ab.........B Dns:[X.........] Ins:[X.........] Vol:[X.........]
9  : -.........B Dns:[X.........] Ins:[X.........] Vol:[XXXXXX....]
Percentage of instances presented(Dns): 100%
```

Figure 5.12: Bin list with information about class distribution.

it is useful to have small values emphasized in the output. It can also be important to show if a bin is completely empty, so an empty bin is represented with ten space characters.

In the last line, the value after `Percentage of instances presented` refers to the density column and is the sum of all instances represented by all `X` characters. Reporting this value should help to detect a distortion of the histogram—in case one of the bins is very narrow and its density value is very large. In relation to very dense bins, other bins get very low density values and become invisible, hiding features of the dataset. Then Multi-TUBE will have to be rerun with different values for parameters like minimal bin width, to avoid the distortion.

As a user parameter, the number of characters can be increased so that for example, twenty `X` characters stand for 100 percent, allowing the output to show more detail. Further features could be added with a fully graphical interface and some ideas for future features are summarised in the later section 'Future work' (Section 5.5.6).

5.5.3 Bin Lists For Two-Class Problems

For a classification dataset with a binary class, the distribution of the classes in the different bins can also be of interest. With an additional column the distribution of a binary class (with two possible values, i.e. 'true' and 'false') can be documented in the bin list. This is shown in Figure 5.12.

The representation is again very coarse in order to give a quick overview. The first ten characters after the first column in each row give the density of

```
#5:
[XXXXXXXXXX][XXXXXX....]
  [XXX.......][...XX.....]
    [XXXXXXXXXX][XXXXXXXXXX]

#6:
[XXXXXXXXXX][XXXXXX....]
  [XXX.......][.....XXXXX]
    [XXXXXXXXXX][XXXXXXXXXX]
```

Figure 5.13: Positions of Bin 5 and Bin 6.

the first and second class, each as a sequence of characters. The sequence for the class with the lower density is hiding the lower part of the sequence for the second class. For example the sequence `[-aaaaab....-]` stands for a bin with class `a` at 50 percent and class `b` at 60 percent. 100 percent is the highest percentage of all bins and both classes for this column.

The characters before and after this character sequence have special meaning. The character after the sequence, given as an uppercase `A` or `B` shows which of the two classes was denser in this bin. This is important if the two classes have the same number of characters in the middle part. The character before the string indicates which of the classes was represented zero times in a bin, or is written `0` if neither was. When no instances at all are in a bin, the bin is represented by the following string: `[-..........-]`.

Note that any bin list output after the mixing of two binning operations gives the bins in the order from highest difference density to lowest difference density (Section 5.5.1).

A graphical user interface would make it possible to draw blocks of exact length. This would show more detail but may also obscure higher level features of the data that the user needs to discover at an early point of exploration.

5.5.4 Bin Position Overview

The *bin position* representation is designed to give an overview of where in the total attribute range a bin is positioned. Figure 5.13 gives two examples of a *bin position overview*: for Bin 5 and Bin 6 from the bin list in Figure 5.10. For each attribute, a string of characters shows the part of the full range the bin covers. If the string is `[XXXXXXXXXX]`, the bin was not cut in this attribute and covers its whole range. The string `[XXXXX.....]` means the range was cut in

approximately the middle of the range and the bin covers the first part of the range. If the bin is very slim (below 0.1 percent of the range), an `I` character is given instead of an `X`.

In the given example, the split tree never selected a cut on the last two attributes, so these attributes could be ignored for the presentation. The selection of attributes for presentation can be important if the number of attributes is very high.

5.5.5 Examples of Data Exploration

In this section two datasets are explored by examining their bin lists, which represent their multidimensional histograms, and for some bins their bin position overviews. Both datasets are two-class datasets with the class values 'positive' and 'negative'. The datasets are also used for experiments in the application multiple-instance learning, discussed in the next chapter (Section 6.2).

Both datasets explored in this section are binary and therefore well suited to demonstrate the 'Mixing of Two Binnings' method introduced in Section 5.4.1. To summarise, this method splits the dataset into two subdatasets, one dataset with all negative instances and one with all positive instances. In the examples given in this section, each of the datasets was first split a certain small number of times using Multi-TUBE binning considering the combined maximum range of the attributes. Following that, the two binnings were mixed and both datasets filled into the binning to obtain a density model.

For each example histogram, the highest mode is indicated first in the output. As the histogram was constructed using the mixing of binnings technique, the mode bin is the maxima of the difference of the density function. The density column ('Dns:') always refers to the normal density of the bin. As additional information each bin list gives as the last value in each row the exact percentage of instances in the corresponding bin.

Both datasets used in this section have the class values 'negative' and 'positive' and with that the letters `A` and `a` stand for the negative instances and `B` and `b` for the positive instances.

The *eastwest* Dataset

The eastwest dataset [6] was first split into fifteen bins using Multi-TUBE. (This example does not use the mixing of binning process.) Fourteen of the 26 attributes were cut. The bin list in Figure 5.14 shows that the bin with the

```
Bin List - with instances percent:
Highest Mode Bin: 3

0 :0b.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<0.0001...] 6.57%
1 :0a.........B Dns:[<0.1......] Ins:[X.........] Vol:[<1E-10....] 8.92%
2 :0b.........A Dns:[X.........] Ins:[X.........] Vol:[<1E-12....] 8.45%
3 :0aaaaaaaaaaB Dns:[XXXXXXXXXX] Ins:[XXXXX.....] Vol:[<1E-13....] 44.6%
4 :-..........- Dns:[          ] Ins:[          ] Vol:[<1E-11....] 0.0%
5 :0b.........A Dns:[<0.01.....] Ins:[X.........] Vol:[<1E-9.....] 5.16%
6 :0b.........A Dns:[<0.001....] Ins:[X.........] Vol:[<1E-8.....] 7.04%
7 :0a.........B Dns:[<0.0001...] Ins:[X.........] Vol:[<0.000001.] 7.98%
8 :-..........- Dns:[          ] Ins:[          ] Vol:[<0.00001..] 0.0%
9 :Ab.........B Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.001....] 3.76%
10:Ab.........B Dns:[<1E-10....] Ins:[X.........] Vol:[<0.01.....] 3.29%
11:0b.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.1......] 2.35%
12:-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
13:-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
14:Ab.........B Dns:[<1E-14....] Ins:[X.........] Vol:[XXXXXXXXXX] 1.88%
```

Figure 5.14: Eastwest dataset: fifteen bins bin list; two-class problem.

highest density is Bin 3, with 44.6 percent. This bin only has a small volume with less than $1E-13$ percent of the total volume.

Most bins are mixed with negative and positive instances in similar densities. The `A` in the first column of the first column block for the Bins 9, 10 and 14 indicates that they contain no negative instances. Bin 14 is the only bin with a very high volume (more than 90 percent of the total volume) and it contains less than 10 percent of the instances. This is thus an example of an aspect of the curse of dimensionality, that most of the high dimensional range is empty or almost empty.

```
#3:
[XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX][XX........][XX........]
  [XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX][.........I][XX........]
    [.........I][XX........][XX........][XX........][.........I]
      [XX........][XX........][XXXXXXXXXX][XX........][XX........]
        [XXXXXXXXXX][XXXXXXXXXX][XX........][XXXXXXXXXX][XXXXXXXXXX]
          [XXXXXXXXXX]
```

Figure 5.15: Bin 3: bin position overview; eastwest dataset.

Bin 3 is the bin with the most instances but with very small volume. Its bin position overview in Figure 5.15 shows that 14 attributes have been cut for this bin. For example, the attributes 1, 2 and 3 have not been cut at all and therefore are shown as `[XXXXXXXXXX]`; the attribute 4 has been cut between 10 and 20 percent of the total range and the lower values are part of the bin.

```
#14:
[XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX][..XXXXXXXX][XXXXXXXXXX]
  [XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX]
    [XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX]
      [XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX]
        [XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX][XXXXXXXXXX]
          [XXXXXXXXXX]
```

Figure 5.16: Bin 14: bin position overview; eastwest dataset.

The attributes 11 and 15 have been cut into very slim bins so that an `I` is shown for them. It can be seen on this output that Bin 3 is positioned in the range of low values.

The bin position overview of the Bin 14 (in Figure 5.16), the bin with the largest volume, shows that only one attribute (attribute 4) has been cut for this bin, and that most of the range is covered for this attribute.

In a second experiment the eastwest dataset was split again using the method of mixing of two binnings. In this case, both subdatasets were cut into 5 bins and the mixing of these two binnings resulted in 36 bins (Bin 0 to Bin 35). Only 6 of the 26 attributes were cut. The highest mode bin according to the difference of density values (positive - negative density) is Bin 0, which has the highest proportion of positive instances compared to all other bins but only 3.76 percent of all instances.

In Figure 5.17 is the bin list of the resulting multidimensional histogram. The bins are ordered by their difference density (as always after the mixing of two binnings). The bins are not completely listed, because bins 3 to 33 look very similar, and all of these bins are empty. In this dataset only 5 bins of 36 are non-empty. None of the non-empty bins have large volume.

The bins 0, 1 and 2 all have no negative instances and therefore an `A` is shown in the first column of the first block of columns. All three bins have very small volume—smaller than 0.1 percent and only a small number of instances—3.76 percent, 3.76 percent and 1.88 percent respectively.

The bins 34 and 35 have positive and negative instances, but more negative instances than positive ones, therefore the `A` in the last column of the first block. Bin 35 has 88.73 percent of the instances but is also a small bin. The number of positive and negative instances in this bin is similiar.

For reference, a bin list generated with standard Multi-TUBE for 36 bins (not shown) had 14 non-empty bins instead of only 5.

```
Bin List - with instances percent:
Highest Mode Bin 0

0 :Ab.........B Dns:[X.........] Ins:[X.........] Vol:[<0.001....] 3.76%
1 :Ab.........B Dns:[X.........] Ins:[X.........] Vol:[<0.001....] 3.29%
2 :Ab.........B Dns:[<0.01.....] Ins:[X.........] Vol:[<0.1......] 1.88%
3 :-..........- Dns:[          ] Ins:[          ] Vol:[<0.01.....] 0.0%
...
19:-..........- Dns:[          ] Ins:[          ] Vol:[XXXXXX....] 0.0%
...
34:0b.........A Dns:[<0.001....] Ins:[X.........] Vol:[<0.01.....] 2.35%
35:0bbbbbbbbbaA Dns:[XXXXXXXXXX] Ins:[XXXXXXXXX.] Vol:[<0.001....] 88.73%
```

Figure 5.17: Eastwest dataset: mixing of binnings bin list (started with 5 bins).

The *elephant* Dataset

The elephant dataset [6] was split with Multi-TUBE into fifteen bins and the resulting histogram is represented as a bin list in Figure 5.18. Only 14 of the 231 attributes have been cut. The result does not show any too narrow cuts. In Bin 0, a high percentage of instances is found, namely 87.2 percent. Most bins are mixed with negative and positive instances in similar densities.

The `A` or `B` in the first column of the first column block shows that several bins have either no negative or no positive instances in them. Bin 14 is the only bin with very high volume—more than 90 percent of the total volume—and it contains less than 0.5 percent of the instances.

```
Bin List - with instances percent:
Highest Mode Bin 0

0 :0bbbbbbbbbbA Dns:[XXXXXXXXXX] Ins:[XXXXXXXXX.] Vol:[<1E-25....] 87.2%
1 :0a.........B Dns:[<0.1......] Ins:[X.........] Vol:[<1E-24....] 1.44%
2 :Ab.........B Dns:[<0.001....] Ins:[X.........] Vol:[<1E-22....] 0.58%
3 :Ab.........B Dns:[<0.00001..] Ins:[X.........] Vol:[<1E-20....] 0.29%
4 :0a.........B Dns:[<1E-8.....] Ins:[X.........] Vol:[<1E-18....] 1.44%
5 :0b.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<1E-16....] 1.73%
6 :Ab.........B Dns:[<1E-12....] Ins:[X.........] Vol:[<1E-14....] 0.93%
7 :0a.........B Dns:[<1E-14....] Ins:[X.........] Vol:[<1E-13....] 0.65%
8 :0b.........A Dns:[<1E-16....] Ins:[X.........] Vol:[<1E-11....] 1.37%
9 :Ba.........A Dns:[<1E-18....] Ins:[X.........] Vol:[<1E-9.....] 0.29%
10:Ab.........B Dns:[<1E-21....] Ins:[<0.1......] Vol:[<0.00001..] 0.07%
11:0..........B Dns:[<1E-21....] Ins:[X.........] Vol:[<0.001....] 1.58%
12:0..........B Dns:[<1E-23....] Ins:[X.........] Vol:[<0.1......] 1.73%
13:Ab.........B Dns:[<1E-26....] Ins:[X.........] Vol:[X.........] 0.22%
14:Ba.........A Dns:[<1E-28....] Ins:[X.........] Vol:[XXXXXXXXXX] 0.5%
```

Figure 5.18: Elephant dataset: fifteen bins bin list; two-class problem.

```
Bin List - with instances percent:
Highest Mode Bin 0

0 : 0a.........B Dns:[<0.1......] Ins:[X.........] Vol:[<1E-13....] 2.73%
1 : 0a.........B Dns:[<0.1......] Ins:[X.........] Vol:[<1E-13....] 1.65%
2 : 0a.........B Dns:[<0.01.....] Ins:[X.........] Vol:[<1E-13....] 0.5%
3 : 0a.........B Dns:[<0.1......] Ins:[X.........] Vol:[<1E-13....] 0.93%
4 : 0a.........B Dns:[<0.001....] Ins:[X.........] Vol:[<1E-11....] 2.52%
5 : Ab.........B Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-11....] 0.14%
6 : Ab.........B Dns:[<0.0001...] Ins:[<0.1......] Vol:[<1E-11....] 0.07%
7 : Ab.........B Dns:[<0.000001.] Ins:[X.........] Vol:[<1E-9.....] 0.43%
8 : Ab.........B Dns:[<1E-11....] Ins:[X.........] Vol:[<0.001....]  0.22%
9 : -..........- Dns:[          ] Ins:[          ] Vol:[<1E-11....] 0.0%
...
74 : -..........- Dns:[          ] Ins:[          ] Vol:[XXXXXXXXXX] 0.0%
75 : Ba.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.001....] 0.22%
76 : 0b.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.00001..] 0.65%
77 : 0b.........A Dns:[<0.001....] Ins:[X.........] Vol:[<1E-11....] 2.8%
78 : Ba.........A Dns:[<0.01.....] Ins:[X.........] Vol:[<1E-13....] 0.29%
79 : 0bbbbbbbbbbA Dns:[XXXXXXXXXX] Ins:[XXXXXXXXX.] Vol:[<1E-15....] 86.84%
```

Figure 5.19: Elephant dataset: mixing of binnings bin list (started with 5 bins).

The elephant dataset was again split using the mixing of two binnings method to examine the difference of the densities between the two classes in the dataset. The first splitting of both subdatasets (positive and negative) into 10 bins resulted in a histogram of 3584 bins after the mixing of the binnings, of which less than 40 were non-empty.

A second split with only five bins per subdatasets resulted in a total of 80 bins. The partial bin list for this histogram is shown in Figure 5.19. Eight of the 231 attributes were cut. Eight bins have more positive instances than negative instances (letter `B` in the last column of the first block) and Bin 0 is the highest mode of the difference density with the highest relative density of all bins. Bin 79 contains 86.84 percent of all instances but is a very small bin and would thus be a mode bin according to the joint density of the two classes. The positive and negative instances in this bin are similarly distributed.

As in the previous example the mixed of binnings method appears to be better able to detect empty areas. For reference, a bin list generated by running Multi-TUBE on the union of the two classes, with 80 bins (not shown) had 33 non-empty bins instead of only 14.

Summary: Data Exploration

In a multidimensional range the instances are generally more concentrated in certain areas, leaving large areas empty as explained by aspects of the 'curse of dimensionality' phenomenon. The illustrative results given above indicate that the mixing of two binnings technique can find empty areas better than a simple Multi-TUBE binning on the full dataset. The bin lists of the generated histograms show whether the instances are spread out or are very concentrated and whether areas with only one class value can be found.

5.5.6 Future Work: Graphical User Interface Support

As an avenue for future work, it would be useful to combine the above methods with flexible functionality and a full graphical user interface to build an effective tool for the support of the data exploration task. This section describes possible GUI support for the representation method considered above.

Both new visualization methods, the *bin list* and the *bin position overview* represent the data in a very coarse fashion. A future feature could be to give more precise values for the data in addition to the coarse grained output. A mouse click on one of the bins could open a new window with this more precise information. Another way would be to give optional view settings that can easily be changed, perhaps a scale setting in a text file that changes the zoom level of the text.

Several graphics could be combined with each other: if a bin is selected in one graph it is also highlighted in the other. This could also be combined with a matrix plot where the ranges of the selected bin are drawn into the scatter plots.

Furthermore, a hierarchical nesting of binnings could be enabled. In this case a bin could be selected and the histogram algorithm could define a histogram with a given number of bins using the instances in that bin only. The clustering algorithm described in the next chapter in Section 6.1 could be used to identify density modes within the bin.

5.6 Empirical Evaluation Using Multiple Instance Learning

To extend the empirical evaluation of Multi-TUBE the Multi-TUBE's clustering method is applied to multiple instance learning (both described in the next chapter, Section 6.2). The multidimensional clustering process used in conjunction with mixed binnings is used to improve the runtime behaviour of a widely known multiple instance learning method.

To provide an indirect, informal evaluation, the above methods for representing Multi-TUBE's binning will be used to document the results of the application of Multi-TUBE to multiple-instance learning in the next chapter and explain some of them with the information gained about the structure of the data.

5.7 Summary

Multi-TUBE is an adaptation of TUBE to multidimensional data. The basic parts of the Multi-TUBE algorithm are the same as in univariate TUBE. Multi-TUBE has all attributes to consider when splitting, and it does so with each of them independently inducing axis-parallel cuts. Again all splits are selected using the likelihood criterion, with the difference being that Multi-TUBE's calculation of the density function differs in the use of the volume of a bin instead of simply the width of a bin.

The multidimensional binning algorithm performs splits recursively and builds a binary density estimation tree. Each cut splits a multidimensional range into two multidimensional rectangular bins in an axis-parallel fashion. For each resulting bin a locally optimal split is computed and added to a priority queue. The next split to be performed is taken from the priority queue, which is ordered according to likelihood gain. The size of the tree is controlled using the same methods as in the one-dimensional TUBE algorithm: the cross-validated log-likelihood criterion and the local minimum description length criterion.

The binning produced by the one-dimensional TUBE method can be represented as a histogram, a method traditionally used for data exploration. Multi-TUBE's binning consists of multidimensional hyperrectangles, which each have a corresponding density value. A presentation technique, called bin list, has been developed to present these 'unordered' bins in a way that makes it pos-

sible for a user to gain insight into the structure of the data. The diagram methods developed are not restricted by the number of instances in the data and can also be applied to very large datasets.

In the next chapter Multi-TUBE is applied in two applications: the TUBE clustering algorithm and multiple instance learning. For the latter application a method of mixing two binnings was implemented as described in this chapter. It is used to construct the difference of the density functions of the positive and the negative instances, which occur in multiple instance learning. The evaluation of Multi-TUBE should demonstrate whether it implements a multidimensional density estimation technique that successfully avoids the problematic aspects of the curse of dimensionality.

Chapter 6

Applications of Multivariate Density Estimation Trees

This chapter discusses two applications of the multidimensional tree-based density estimator Multi-TUBE. The chapter is split into two corresponding parts, the first section (Section 6.1) is about the clustering with Multi-TUBE's binning, and the second part (Section 6.2) discusses the application of TUBE clustering to multiple-instance classification.

Multi-TUBE splits one attribute alone in each split, and in this way cuts the range in an axis-parallel fashion. The result of Multi-TUBE's binning are multidimensional rectangles, which represent areas exhibiting approximatly uniform distribution.

Clustering has been applied in statistics, pattern matching and data mining. As Berkhin points out in his survey of clustering data mining techniques [8], the application to data mining has the added requirement of being able to process large datasets with often many attributes. Using Multi-TUBE's binning, a TUBE clustering algorithm was developed that is capable of processing large datasets with high dimensionality. This algorithm is presented in Section 6.1.

Multiple-instance learning is the second application in this chapter. In multiple-instance learning, each example consists of a bag of instances. In a positive bag only a few of the instances are generally considered to be 'real' positive instances. In the standard multiple-instance scenario, the remaining instances in a positive bag are assumed to be noise. Maron's Diverse Density algorithm [43] uses a probability measure to find the positive concept areas. In this thesis a multiple-instance classifier is presented which defines the positive concept area using TUBE clustering. In addition the TUBE clusters are also

used to improve the runtime behaviour of Maron's Diverse Density algorithm. The multiple-instance data application is covered in Section 6.2.

6.1 Application: Clustering

Clustering is the grouping of instances according to some similarity measure. Finding clusters in data can be important if the task is to find concepts that can be used to label instances. To do this, first clusters are found, then each concept is assigned to one or more clusters. The work in this thesis is on multidimensional datasets consisting of numeric attributes (and no nominal attributes) and so the instances can be seen as distributed in multidimensional space. A cluster comprises instances clustering close together and so forming an area of high density. Since clusters can have various shapes, their surrounding area of lower density is important for the definition of a cluster's boundary. The surrounding low density area also forms a border against other clusters.

The simplest form of a cluster is a point. In real-world datasets it is rare for a significant number of instances to have attributes with identical values to form a point-like cluster. More likely, the instances are distributed around a point-like centre. The area of such a cluster forms a multidimensional ball. The distribution of the instances in the cluster can, for example, be the normal distribution. It forms an ellipsoid if the variations of the attributes differ. Between clusters, the area is not necessarily empty because there can be instances that resulted from noise in the data or other outlier values that do not fit into any concept.

The TUBE binning method can be used to develop a clustering algorithm that finds clusters of varying shapes in the data. The TUBE clusterer proposed in the following is a mode-seeking clustering algorithm. In fact it is actually a bump-hunting method according to the terminology of Silverman [60] because its emphasis is on detecting clusters in sets of data rather than on finding modes in underlying densities. Silverman discusses equal-width binning and points out the indirect correlation between the width of a bin and the number of modes found. The TUBE binning method builds a multidimensional histogram describing the density in each bin and the multidimensional blanket of values represent an estimation of the underlying density function. The more the area is split into bins, the greater the irregularities in the resulting function. With fewer bins, the function will be smoother and will show fewer modes. Cutting the area into more bins reveals more details of the distribution and more modes can be found.

The concept of a cluster is generally only defined indirectly by the method of cluster analysis being used. This fact makes it difficult to compare two clus-

tering algorithms. Nevertheless, this section empirically compares the TUBE clusterer with clustering using EM-based mixture modelling and points out the differences between the cluster structures found.

In Section 6.1.1, the clustering algorithm is explained in detail. Section 6.1.2 discusses some problems and parameter settings that can impact on the clustering process. Section 6.1.3 introduces as related work clustering algorithms with similarities to the TUBE clusterer. Section 6.1.4 evaluates TUBE's ability to find an appropriate number of clusters by comparing it to cross-validated EM (EM-CV), an algorithm that also finds the number of clusters automatically. The type of clusters the two algorithms find are also compared. Section 6.1.5 summarises this section about the TUBE clusterer. Note that the TUBE clustering algorithm is also used in the multiple instance application, which is discussed in Section 6.2.

6.1.1 The Multidimensional TUBE Clusterer

The binning produced by the multivariate TUBE algorithm provides a good basis for a range of clustering methods. It supplies an estimation of the density distribution of the data and therefore makes it easy to determine the modes in the distribution. Hence, a *mode-seeking* clustering algorithm was developed and implemented. The algorithm was named *TUBE clusterer* and can be classified not only as a *density-based* clusterer but, since the multidimensional histogram forms a probability function, also as a *probability-based* clusterer.

What is a Mode? A mode is a peak or bump in the multidimensional density function that is an area of high density surrounded by an area of lesser density—a local maximum. As a first step, the algorithm performs a binning on the data, then it finds the modes among the bins and finally forms clusters by combining the mode bins with their surrounding lower-density bins down to the 'valleys' of the distribution. Silverman [60] calls a 'bump' in a density function an interval $[a, b]$ such that f is concave over $[a, b]$ but not over a larger interval. Two questions that remain are whether the entire bump should be taken as cluster and what should happen to the valley bins.

An Illustrative Example Figure 6.1 gives a very simple example of how the TUBE clusterer works. The example uses one-dimensional data, which was generated from two different normal distributions. The range of the values was

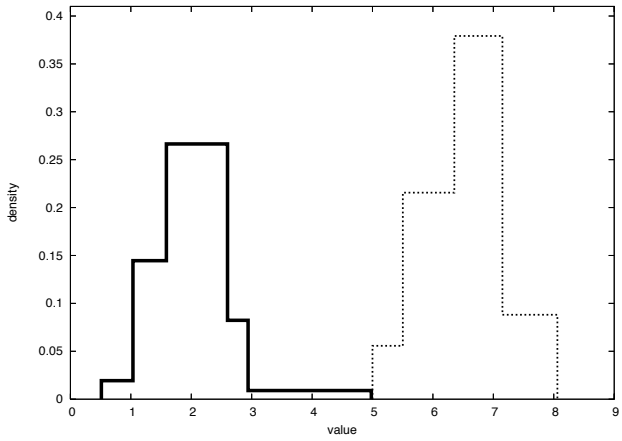

Figure 6.1: TUBE clusterer found two clusters.

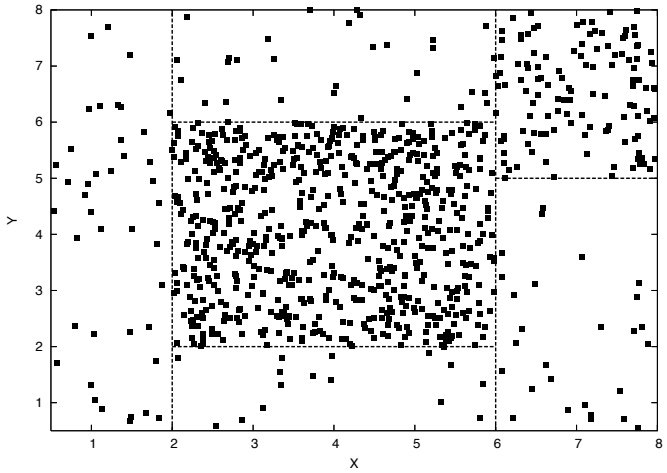

Figure 6.2: Bin has three neighbours in the first attribute X.

discretized using the TUBE discretization algorithm. The TUBE clusterer was used to define clusters on the resulting bins. The algorithm found two modes and defined two clusters around them by gathering all bins around each mode that have lower density until reaching a valley of the density function. A valley bin is a bin that is surrounded by bins of the same or higher density. In this case the one valley bin between the modes was assigned to the left cluster.

Finding the Modes Each binning forms a histogram, which is a smoothed representation of the distribution of the data and an estimation of the real unknown distribution. The TUBE binning method selects the bin width according to the underlying density, so it is plausible that the smoothing factor is chosen appropriately. If this is so, then the modes (or 'peaks') found in the histogram correlate well with the modes of the real unknown distribution of

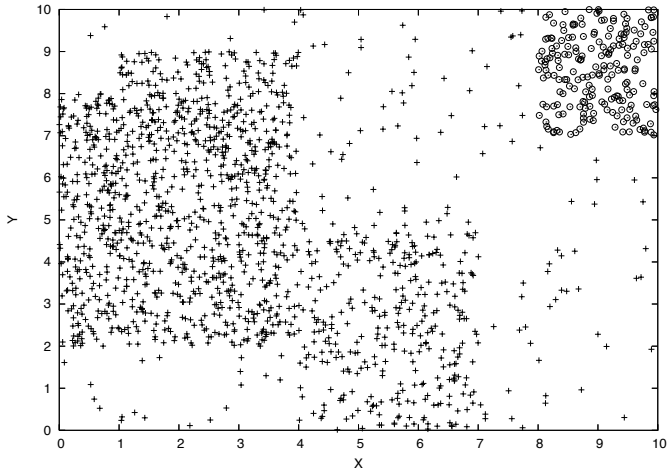

Figure 6.3: TUBE finds two cluster. Cluster1 is plotted with crosses, Cluster2 with circles.

the data.

In a one-dimensional dataset, finding a peak in the list of bins can be done with little effort. The mode searching algorithm considers, for each bin, the density of both neighbours. Each bin for which both neighbouring bins exhibit lesser density is a mode bin. In the multidimensional case, the bins do not have any order defined on them. Additionally, each bin can have several neighbours and maybe even more than two in one direction. As an example the bin in the middle of the scatterplot in Figure 6.2 has three neighbours in the direction of X. The mode-seeking algorithm that was developed for multidimensional data is explained in detail below.

Finding Outliers and Ignoring Noise There are different ways of interpreting the data's structure and detecting areas relevant to the target concept and areas where the instances should be discarded as noise or should be defined as outliers. It is reasonable to assume that areas of the same density should always be defined as either concept areas or empty areas that contain some noise. The TUBE-binning algorithm looks along an axis-parallel line for a significant change in the density level and cuts the bins, starting from the whole range of the attribute values, recursively into smaller bins. As a result, each of the bins is a multidimensional axis-parallel rectangle that spans an area of similar density. Using user input (e.g. a density threshold value) the clustering algorithm can then consider each bin and decide if it is part of a concept or part of 'empty' area (which would mean the instances in it should be viewed as noise or outliers).

See Figure 6.3 for how the TUBE clusterer defines clusters in a two-dimensional dataset. The low density area (valley bins) has been assigned to Cluster1 but could be discarded as noise using appropriate user parameter settings.

Representation of the clusters Since the clusters generated by the TUBE clusterer consist of multiple multidimensional axis-parallel rectangles, a cluster can be represented as conjunctive statements combined with disjunctions. For this representation, $min_{i,j}$ is the minimum value of attribute i in bin j, $max_{i,j}$ is the corresponding maximum value of attribute i in bin j, and $a_1, a_2..a_i..a_k$ are the attribute values of an instance. A cluster consisting of m bins can then be represented as:

$(a_1 > min_{1,1} \wedge a_1 < max_{1,1}) \wedge .. \wedge (a_k > min_{k,1} \wedge a_k < max_{k,1})$
$\vee (a_1 > min_{1,2} \wedge a_1 < max_{1,2}) \wedge .. \wedge (a_k > min_{k,2} \wedge a_k < max_{k,2})$
...
$\vee (a_1 > min_{1,m} \wedge a_1 < max_{1,m}) \wedge .. \wedge (a_k > min_{k,m} \wedge a_k < max_{k,m})$

Algorithm to Find Modes in the Multidimensional Binning

As mentioned above, the TUBE clustering algorithm requires the modes of the estimated density to be found. An important part of TUBE's mode-finding algorithm is to determine if two bins are neighbours. It turns out that the neighbourhood test can be made more efficient by reducing the number of attributes used for the test. The mode-finding algorithm and the neighbourhood test are introduced in detail in the following.

A Simple Mode-Finding Algorithm A straightforward way to find mode bins is to check, for each bin, whether any of the other bins are adjacent to it and have greater density. If no such neighbour is found, the bin is declared to be a mode. But, as soon as one denser neighbour is found, it is clear that the bin cannot be a mode and no further bins have to be tested. So, in the worst case, for each bin all other bins have to be tested: For all N bins check if the other $N - 1$ bins are neighbours. This means that the worst-case time complexity of this test is $O(N^2)$.

A Faster Mode-Finding Algorithm The following change to the algorithm reduces the work that needs to be done to find all mode bins. As preparation, all bins are sorted according to their density value. The densest

bin can be immediately declared a mode bin because there is no denser bin it can be adjacent to. Starting from the second densest bin, all the remaining bins are tested to see if they are modes. The fact that a less denser bin is a neighbour is irrelevant. Therefore only the denser bins are tested. As soon as one of the denser bins is found to be a neighbour, the bin cannot be a mode itself and the test for this bin can be stopped. Assuming that the number of mode bins is k and $k \ll N$ (N is the total number of bins), the complexity of this algorithm is $O(kN)$. See Algorithm 4 for the pseudocode for this algorithm.

Algorithm 4 Pseudocode: Faster mode-finding algorithm.
Uses Algorithm 5 to test if two bins are neighbours.

```
bins; list of all bins
modeList; empty list of mode bins

{sort all bins from densest to least densest}
sortedBins ⇐ sortWithDensityDecending(bins)

modeList ⇐ add densest bin  {densest bin is always a mode bin}
for bin_i starting from second-densest bin to last bin in sortedBins do
  bin_i is a mode ⇐ true

  {check all denser bins if neighbour of bin_i}
  for bin_j from densest bin to bin_{i-1} in sortedBins
  and while bin_i is a mode do
    if bin_i is neighbour of bin_j {← see Algorithm 5} then
      bin_i is a mode ⇐ false  {found denser neighbour}
    end if
  end for

  if bin_i is a mode = true then
    modeList ⇐ add bin_i  {found a mode bin}
  end if
end for
return modeList
```

Two Bins Neighbourhood-Test The test used to see whether two bins are neighbours can be supported by the tree structure that was constructed when cutting the bins (the *split tree* generated by Multi-TUBE). In each node, the split tree tells us which attribute was cut at which value. The leaf nodes are the bins. In the implementation presented here, each bin holds the information

about the split path to its leaf node. The information about a split can be used to determine if two bins are neighbours. For example, if one bin's split history contains the split ($a_i < 3$) and the second bin has in its split path the condition ($a_i > 5$) then it is clear that these two bins are not neighbours. In practise, instead of analyzing the split conditions, the TUBE clusterer tests if the subranges of the attributes overlap or share one point. The pseudocode in Algorithm 5 details the neighbourhood test. In the next paragraph, the details of the attribute selection step it employes are explained.

Algorithm 5 Pseudocode for Test if bin_a and bin_b are neighbours
Uses Algorithm 6 to select the set of attributes to be tested.

```
Given a dataset with k attributes;
Each bin_i has a range ([min_{i,1}, max_{i,1}], ..[min_{i,k}, max_{i,k}])

testFlags; boolean array of same length as number of attributes
testFlags ⇐ findTestAttributes()  {← see Algorithm 6}

binsOverLap ⇐ true
for each attribute i and while binsOverLap do
  if testFlags[i] = true then
    if min_{a,i} > max_{b,i} then
      {considering attribute i, bin_a is to the right of bin_b}
      binsOverLap ⇐ false  {and can finish testing}
    else
      if min_{b,i} > max_{a,i} then
        {considering attribute i, bin_b is to the right of bin_a}
        binsOverLap ⇐ false  {and can finish testing}
      end if
    end if
  end if
end for
```

Reducing the Set of Attributes in the Neighbourhood Test Since TUBE clustering is designed to be applied to datasets that can have a large number of attributes, it is important to reduce the number of attributes to be tested. It is obviously sufficient to test only the attributes that have been cut. The depth of the split tree is most likely much smaller than the number of instances, meaning not all attributes have been split to form the bins.

To further reduce the number of attributes that need to be tested for overlapping subranges, the split paths of the two bins being compared are analyzed. A bin's split path is a string describing the path of the bin from the root of the

split tree down to the leaf where the bin is located. It contains 'L' and 'R' characters for the way *left* or *right* down the split tree (eg. $LLRL$ or $RLRLLL$). The common prefix of the two paths can be ignored. As an example, assume bin_k has the split path $LLRL$ and bin_l has the split path $LLRRRR$. The common prefix is LLR, which stands for the path $left-left-right$. The two bins follow this part of the path simultaneously starting from the root. Before the fourth split bin_k and bin_l are actually one bin. The next split cuts them apart, with bin_k being the left bin and bin_l being the right bin. The right bin is split further. The next two cuts on the right bin are important to check to see if the resulting bin bin_l is still a neighbour of bin_k. As soon as the split paths of the bins diverge, all attributes in both further split histories are gathered. This set of attributes is not the smallest possible selection of attributes that need to be tested, but one that can be achieved with little computational effort. See Algorithm 6 for the pseudocode detailing this method.

Collecting Bins to Form Clusters At the same time as the search for modes is executed the bins can be assigned to clusters. Each non-mode bin is normally assigned to the same cluster as a denser neighbour that was found when testing if the bin is a mode bin itself. Algorithm 7 is the same as Algorithm 4 but with the commands added that are used to gather the bins into clusters. (These commands are labelled with {**}.) Each mode corresponds to one cluster.

For valley bins, it is not clear which cluster they should be part of. Depending on the application two scenarios are possible: The valley bins are assigned to neither cluster or each valley bin is assigned to every neighbouring cluster. For the second solution the algorithm has to be repeated for each mode bin to find all clusters each valley bin is in.

Furthermore, it can be useful to set a minimum density threshold for clusters. Bins with a threshold below this threshold are considered not to be part of any cluster and are stripped from the cluster bin list in a post-processing step.

6.1.2 Discussion of the Algorithm

Cluster analysis has been applied to many areas like medicine, the social sciences and biology as part of data mining, statistics and pattern recognition [31]. From these applications an immense number of clustering algorithms have ap-

Algorithm 6 Pseudocode: **function** findTestAttributes;
Selects the attributes to be tested

```
path_a; character string containing split path of bin_a
path_b; character string containing split path of bin_b

function stepDownTree (node, index)
if path_a[index] = "L" then
  node ⇐ node.left  {go down the left branch}
else
  node ⇐ node.right  {go down the right branch}
end if
return  node
end function

testFlags; boolean array of same length as number of attributes
set all values in array testFlags to false

index ⇐ 0 {set index to beginning of split path string}
node ⇐ root of tree

{skip over the path parts which the two bins share}
while  path_a[index] = path_b[index] and not end of path_a or path_b do
  index ⇐ index + 1
  node ⇐ stepDownTree (node, index)
end while

{gather attributes used in path_a}
savenode ⇐ node
node ⇐ stepDownTree (node, path_a[index])
while not reached end of path_a do
  testFlags[node.attribute] = true
  node ⇐ stepDownTree(node, path_a[index])
  index = index + 1
end while
{gather attributes used in path_b}
node ⇐ savenode
node ⇐ stepDownTree (node, path_a[index])
while not reached end of path_b do
  testFlags[node.attribute] = true
  node ⇐ stepDownTree(node, path_b[index])
  index = index + 1
end while

return  testFlags
```

Algorithm 7 Pseudocode: Mode-finding algorithm; Same as 4 with commands to select bins into clusters. These commands are labelled with {**}. Uses Algorithm 5 to test if two bins are neighbours.

```
bins; list of all bins
isInCluster; array with as many integer values as there are bins {**}
modeList; empty list of mode bins
numClusters ⇐ 0; number of modes and clusters is 0 {**}

{sort all bins from densest to least dense}
sortedBins ⇐ sortWithDensityDecending(bins)

modeList ⇐ add densest bin  {densest bin is always a mode bin}
for bin_i starting from second-densest bin to last bin in sortedBins do
  bin_i is a mode ⇐ true

  {check all denser bins if neighbour of bin_i}
  for bin_j from densest bin to bin_{j-1} in sortedBins
  and while believe bin_i is a mode do
    if bin_i is neighbour of bin_j {← see Algorithm 5} then
      bin_i is a mode ⇐ false  {found denser neighbour}
      isInCluster[i] ⇐ isInCluster[j]   {**}
    end if
  end for

  if bin_i is a mode bin = true then
    modeList ⇐ add bin_i  {found a mode bin}
    numCluster ⇐ numCluster + 1  {**}
    isInCluster[i] ⇐ numCluster   {**}
  end if
end for
return modeList
```

peared. Jain [31] gives an overview and taxonomy of existing clustering algorithms. Han and Kamber [24] give typical requirements for clustering in data mining. The first part of this section briefly discusses how the TUBE clusterer fits into Jain's taxonomy and how it fulfils these requirements based on the current implementation of the TUBE clusterer, but also mentions some possible variations of it. It then continues with possibilities for future work.

Jain's Taxonomy

The three main parts of Jain's [31] taxonomy are: the method of measuring similarity between instances, how the grouping into clusters is performed, and the abstract representation of the clusters. The next section discusses these points with respect to the TUBE clusterer.

Similarity Measure The TUBE clusterer only implicitly uses a distance measure. The bins are cut in axis-parallel fashion. The bin-width is a Euclidian distance.

Grouping into Clusters A traditional classification of clustering methods is the division into hierarchical grouping and partitional grouping methods [25]. The TUBE clusterer can be seen as a partitional method. The split tree of the TUBE binning method can be used to define a hierarchy on the clusters to obtain a hierarchical grouping. Note that the split tree does not directly correspond to a hierarchy of the clusters. To give an example, a first split might divide the range into an area with low density and an area where all the clusters are found. Then the split tree at this split has one branch with no cluster in it.

The TUBE clusterer does not make any assumption regarding a cluster's shape. Density based clustering techniques like the TUBE clusterer are well suited to detect clusters with complex shapes.

The current TUBE clusterer forms a hard partitioning on the data, but could easily be extended to a fuzzy assignment. This could be done by using the distances to the centre of the mode bin, or by taking the density of the bin containing the instance into consideration. Instances in a bin close to the mode bin but with much lower density could be considered to be less probable members of the cluster around this mode bin.

Representation of the Clusters The clusters can be represented as conjunctions of disjunctions as stated in detail in Section 6.1.1.

Han and Kamber's Requirements

Han and Kamber [24] summarise a list of requirements for general clustering algorithms (see also Table 6.1).

Requirements the TUBE Clusterer Fulfils The TUBE clusterer fulfils many of the requirements listed by Han and Kamber [24]. To test its scalability it is compared with cross-validated EM in Section 6.1.4), on datasets with varying dimensions and sizes. The results show that it generally runs faster especially on high-dimensional datasets. The TUBE clusterer discovers clusters of arbitrary shape. However, the input of domain knowledge could help with its inability to cluster test data that is out of the training data's range. It can deal with outliers and noisy data and is not sensitive to the order in which the input data is given, although it is not incremental. Its ability to find clusters is not affected by high-dimensional data, and its output represents the clusters in an easily interpretable form.

One further noteworthy property of the TUBE clusterer outside Han and Kamber's requirements is that it finds the number of clusters automatically.

Requirements the TUBE Clusterer Does Not Fulfil The TUBE clusterer as presented in this thesis works only for numeric data. Also it cannot cluster test data that is out of the training data's range and does not permit the inclusion of contraints into the clustering process.

Future Work

This section explains variations of the TUBE clusterer that could be considered for future work. Depending on the application or depending on the way the data is structured, the clustering could be done differently.

Valley Bins Valley bins are the bins that have no neighbours that are less dense and are in the 'valley' of the density distribution. The algorithm could vary in the way it decides on which cluster to assign the valley bins to. In the case of fuzzy assignment, valley bins could be assigned to all neighbouring clusters in parts.

Table 6.1: Han and Kamber's requirements for clustering algorithms

Requirement	Fulfilled by TUBE = X
Scalable	X
Finds clusters of arbitrary shape	X
Can deal with outliers	X
Can deal with noise	X
Not sensitive to input order	X
Incremental	
High-dimensional data	X
Interpretable cluster models	X
Other data types than numeric	
Clusters data out of cluster range	
Inclusion of constraints possible	
Finds number of clusters automatically	X

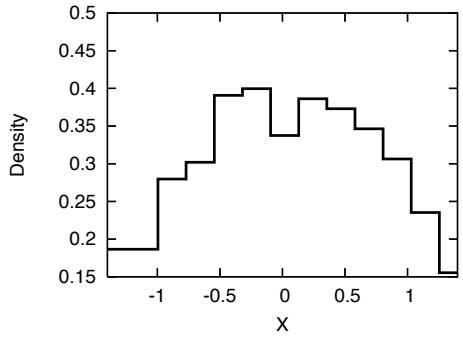

Figure 6.4: Small ridge that splits a cluster.

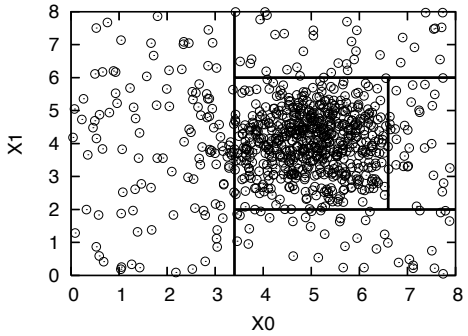

Figure 6.5: Cutting around dense area cuts areas of similar densities apart.

Noise and Outliers Valley bins could be seen as bins with instances that are outliers or noise. Or, more generally, not only valley bins but all bins with density below a certain minimum density level could be excluded from all clusters and classified as noise. This minimum density could be given as a parameter.

Ridges which Split Clusters If the binning is smooth enough, a mode is a good point to define a cluster. However, if the smoothing is not well chosen, two modes found can actually be from the same cluster and can be two peaks connected by a shallow ridge as shown in a one dimensional example in Figure 6.4. The TUBE binning algorithm chooses the smoothing parameter, namely the bin width, automatically. The case of a shallow ridge never oc-

curred in any of the datasets used in the tests performed in this thesis. Still, it cannot be said with certainty that it is avoided by the algorithm. It could be just an arbitrary result of the datasets chosen.

Shredding of the Range If a dense area is located in the middle of the instance space, the area around the dense area is carved out until the dense area is isolated into a bin. This results in splits that divide areas of similar density as can be seen in Figure 6.5. A valuable future task could be to connect the lower density areas around modes to form areas of similar density that embed areas of higher density.

6.1.3 Clusterers with Similarities to the TUBE Clusterer

The TUBE clusterer can be categorized as a *nonparametric density-based* clusterer and also as a *probability-based* clusterer. The TUBE binning method selects the best attribute to cut bins of similar density. Later tests will show that the TUBE clusterer is an effective *subspace* clusterer, a feature very important for clusterers that are designed for high-dimensional dataset. The TUBE clusterer has similarities to grid-based clusterers. However, in the data mining literature no clusterer so far has been classified as *mode-seeking*, which is the strategy implemented by the TUBE clusterer.

Other Subspace Clusterers CLIQUE [2] can also be seen as a grid-based clustering method. It partitions the space into non-overlapping rectangular units. It starts its search for a dense area in one-dimensional space continuing into higher-dimensional space and in this way finds subspace clusters. PROCLUS [1] starts from a high-dimensional space reducing to lower dimensions where it then finds the subspace clusters.

Other Density-based Clusterers DBSCAN [17] and OPTICS [5] grow the area of a cluster by considering the density of instances. DBSCAN examines the neighbourhood of each instance up to a given distance, which is provided as a parameter. If the number of neighbouring instances is larger than a given threshold, the instance is defined as a core for a cluster. Clusters are grown starting from these cores by adding further regions that have a sufficiently dense neighbourhood. Similar to DBSCAN the TUBE clusterer

grows its clusters starting from a mode bin and adds further bins. It has its data summarised in bins (in a similar fashion as grid-based methods) which makes the cluster gathering process efficient, but it does not need the two parameters for the neighbourhood size and the neighbourhood density required by DBSCAN. OPTICS [5] works the same way as DBSCAN, but it tries to overcome DBSCAN's dependency on the two parameters mentioned above and creates an ordering of the instances from which a density-based clustering can be extracted. DBSCAN, OPTICS and the TUBE clusterer can be seen as non-parametric density-based methods in that they do not use a model for their density estimation.

Other Probability-based Clusterers According to Han and Kamber [24] EM is a model-based clustering method and according to Berkhin [8] a probability-based clusterer. Since the model describes a density function it could be seen as a parametric density-based clusterer in contrast to the TUBE clusterer, which is based on a nonparametric density function. EM is used in the evaluation of the TUBE clusterer in the next subsection.

Grid-based Clusterers Grid-based clusterers quantize the instance space and use the grid cells as representatives for the instances in it. The TUBE clusterer also takes its bins as representatives for the instances. The cuts of the TUBE binning do not form a grid but partition the instance space in a tree-like fashion with varying bin width. Like in grid methods, the cuts for the bins are axis-parallel. STING [66] and WaveCluster [58] are well known grid-based clusterers. STING is built for spatial databases and the algorithm constructs a hierarchical grid, which contains statistical information of the data. The statistical data is often sufficient to answer standard queries to the data base. WaveCluster applies a wavelet transform function to the attribute space. WaveCluster is not recommended for datasets with high dimensionality. As mentioned above, CLIQUE also is a grid-based subspace clusterer.

6.1.4 Evaluation of the TUBE Clusterer

Many different clustering algorithms have been invented. Most of them need the number of clusters as input from the user. The TUBE clustering algorithm finds the number of clusters itself. The well known mixture model EM clustering [13] can also automatically decide between models that differ in their

number of clusters. Because it estimates a density function, the cross-validated likelihood can be used for this purpose just like in TUBE. In this section the TUBE clusterer's and EM-CV's ability to find a correct number of clusters are compared. In this comparison, the number of clusters found and also the type of clusters found are analyzed. For this analysis, several datasets were generated. The following sections contain each scenario composed of several tests on datasets using the TUBE clusterer and the EM clusterer.

The EM Clustering Algorithm and cross-validated EM

The EM algorithm The Expectation Maximisation (EM) algorithm is based on finite mixture densities and is a probabilistic clustering method. It assumes that the underlying density of the dataset U is a finite mixture density function f (x_i is an instance of the given dataset, ϕ_j is the set of unknown variables of the finite mixture model and α_j are the weights of the component functions g_j):

$$f^k(x_i|\phi^k) = \sum_{j=1}^{k} \alpha_j g_j(x_i|\phi_j)$$

The algorithm alternates between two steps: finding the unknown variables of the finite mixture distribution and finding the assignment of instances to clusters. Proof of convergence for this algorithm was given by Dempster, Laird and Rubin [13]. The expectation step (E-step) maximises a local lower bound for the posterior distribution. The maximisation (M-step) maximises the posterior probability of the unknown parameters given the data. The lower bound used can be represented as a sum of log's and so avoids the log's of large sums.

Since EM clustering is commonly used with mixtures of Gaussian distributions, EM, like other clustering algorithms that are based on a distance measure, can only find convex, spherical or elliptical clusters [24].

The EM clustering algorithm implemented in the WEKA data mining tool [68] has an option to use cross-validation to select the number of clusters, by cross-validating the likelihood. Note that this implementation also assumes conditional independence of the attributes given the clusters.

Experiments with Generated Datasets

This section discusses the results of experiments that have been done to examine the way TUBE finds clusters in multidimensional data. For these experi-

ments several datasets were generated. The datasets vary in their number of attributes. Some of the generated clusters are subspace clusters, which means that they are clustered over only a subset of the attributes of the dataset and equally distributed over the rest. The first experiments are done on datasets with clusters of spherical shape, with the instances normally distributed around a point. Further test datasets have clusters with oblong shapes and uniform distributions. Last is a dataset that has a cluster with a non-convex shape.

The algorithms used are EM-CV and the TUBE clusterer. Each experiment uses both clustering algorithms and compares the number of clusters found. The shape and the position of the clusters is analyzed and discussed. For these experiments the WEKA machine learning workbench was used [68]. The clustering tab in WEKA has the option to compute and output an error measurement using an extra nominal attribute (this attribute is not used in the clustering process) that gives the true cluster assignment. The error is given as a percentage of the instances clustered wrongly compared to the total dataset. The datasets were generated using the WEKA *Subspace data generator*. Each example dataset has an additional attribute that contains the label for each cluster.

The runtime of the algorithm is part of the results. The time differences between EM-CV and the TUBE clusterer are obvious and therefore were only measured with the coarse measurement method of the Linux *time* command, which gives the total time elapsed and not the time when only the CPU was used.

First, the generated datasets were clustered using EM and the TUBE clusterer. For these experiments the TUBE clusterer had the maximum number of bins set to 100. The valley bins were not assigned to any cluster.

Tables 6.2 etc. give details for the generated datasets that were used. The tables contain, for each cluster, the attributes it was generated in (column 'Attrs') and the number of instances (column 'Insts') that are generated in the cluster. The generated datasets contain two types of clusters, clusters with a Gaussian distribution, for which the mean values and the standard deviations in each dimension are given, and uniform clusters, for which the minimum and maximum values for each dimension are given.

The experiments are structured in six examples and for each example a result overview table is given with the resulting numbers of clusters and the runtime values of all test runs performed

Example 1: Dataset with three clusters in fifteen dimensions, each with three relevant attributes. This dataset has fifteen attributes and three spherical clusters with a Gaussian distribution (see Table 6.2 below) and 600 instances.

Table 6.2: Example 1: Instructions for the data generator

Example 1: 3 Gaussian Subspace Clusters					
	Distribution	Mean	Variance	Attrs	Insts
Cluster-1	Gaussian	2.0, 4.0, 1.0	1.0, 1.0, 1.0	1, 2, 3	200
Cluster-2	Gaussian	8.0, 0.0, 4.0	1.0, 1.0, 1.0	1, 2, 4	200
Cluster-3	Gaussian	10.0, 7.0, 9.0	1.0, 1.0, 1.0	2, 3, 4	200

Table 6.3: Example 1: Results overview

Algorithm	Num. Clusters Found	Error Rate	CPU-time
EM-CV	3	0.0%	$0m51.727s$
TUBE(100 bins)	3	25.2%	$0m3.786$

Table 6.3 shows that the TUBE clusterer is approximately 17 times quicker than EM-CV to finish the task. EM-CV's clustering is absolutely precise with 0.0% errors because the Gaussian distributions produce spherical clusters, which are ideal for EM. The TUBE clusterer makes rectangular cluster models, which do not fit the generated spherical shape well. Hence TUBE's number of incorrectly clustered instances is 25.2%

Example 2: Dataset with three clusters in four dimensions, each with three relevant attributes. This dataset is almost the same as the one above but it has one additional irrelevant (equally distributed) attribute. This dataset also has 600 instances.

The clusters have Gaussian distributions and are subspace clusters in three dimensions and the fourth irrelevant dimension is equally distributed. The dimensions chosen to be relevant vary between the clusters. The means of the clusters are at least double their standard deviation apart. The dataset does not have further attributes that are irrelevant for all clusters.

Table 6.4: Example 2: Results overview

Algorithm	Num. Clusters Found	Error Rate	CPU-time
EM-CV	7	47.3%	$0m50.372s$
TUBE(100 bins)	3	25.2%	$0m1.418s$

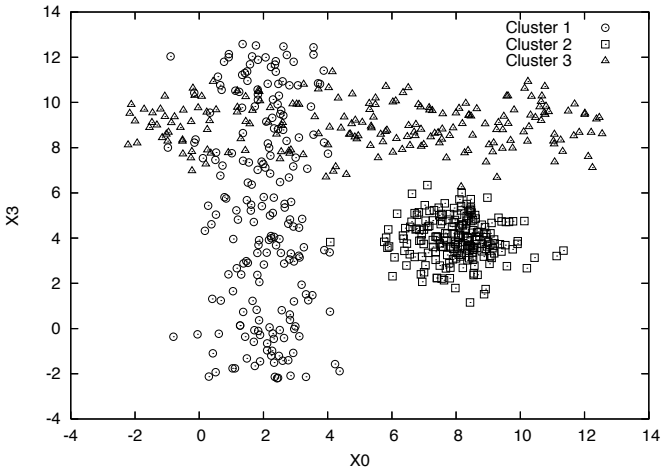

Figure 6.6: Three tube-shaped clusters.

EM-CV finds seven clusters and the TUBE clusterer finds three clusters, which is the same as the number of clusters that were generated. Why does EM-CV find far more clusters? The fourth equally distributed attribute gives the clusters a tube-like shape (see Figure 6.6) and EM-CV sets its normal spherical clusters along that tube. The difference of the number of generated clusters explains the high error rate of 47.3% for EM-CV. The TUBE clusterer makes rectangular shaped cluster models and again this does not fit the generated spherical shape well so 25.2% instances are assigned to a different cluster than the one they were generated in.

The TUBE clusterer is about 50 times quicker than EM-CV.

Example 3: Dataset with eight clusters in four dimensions, with three relevant attributes each. The eight clusters have been generated in a dataset that has four attributes (see Table 6.5). Each cluster is a subspace cluster with only three of the four dimensions relevant. The relevant dimensions vary between the clusters.

The eight clusters each have a Gaussian normal distribution in three dimensions and the fourth dimension is equally distributed. The means are at least ten times the standard deviation apart. When the clusters were set closer the TUBE algorithm only found half the number of clusters.

EM-CV finds all eight clusters. After various generated test datasets that had the eight clusters set closer and TUBE only modelling one or a few clusters, TUBE found seven clusters on this dataset. Analyzing the output of the clusterer shows that cluster two and five have been put into one output cluster.

Table 6.5: Example 3: Instructions for the data generator

Example 3: Eight Gaussian Subspace Clusters					
	Distribution	Mean	Variance	Atts	Insts
Cluster-1	Gaussian	2.0, 14.0, 28.0	1.0, 1.0, 1.0	1, 2, 3	200
Cluster-2	Gaussian	14.0, 28.0, 14.0	1.0, 1.0, 1.0	1, 2, 4	200
Cluster-3	Gaussian	2.0, 2.0, 28.0	1.0, 1.0, 1.0	2, 3, 4	200
Cluster-4	Gaussian	28.0, 14.0, 14.0	1.0, 1.0, 1.0	1, 2, 3	200
Cluster-5	Gaussian	2.0, 28.0, 2.0	1.0, 1.0, 1.0	1, 2, 4	200
Cluster-6	Gaussian	2.0, 28.0, 2.0	1.0, 1.0, 1.0	2, 3, 4	200
Cluster-7	Gaussian	14.0, 14.0, 2.0	1.0, 1.0, 1.0	1, 2, 3	200
Cluster-8	Gaussian	28.0, 28.0, 28.0	1.0, 1.0, 1.0	1, 2, 4	200

Table 6.6: Example 3: Results overview

Algorithm	Num. Clusters Found	Error Rate	CPU-time
EM-CV	8	0.0%	$70m52.239s$
TUBE(100 bins)	7	19.6%	$0m14.612s$

EM-CV again finds the generated clusters precisely with error rate 0.0%.

The TUBE clusterer is 280 times faster than EM-CV to finish this task.

Example 4: Example with diagonally set clusters. TUBE performs axis-parallel cuts. If the clusters are set on diagonals and not on a grid, it should become more difficult for the algorithm to find the clusters. This scenario comprises two datasets that both have only two dimensions. The first dataset has three Gaussian clusters positioned along a row. The clusters have been pushed so far apart that the EM-CV algorithm finds three clusters (see Table 6.7, and Figure 6.7).

Table 6.7: Example 4: Instructions for the data generator

Example 4: Three Clusters in a Row					
	Distribution	Mean	Variance	Attrs	Insts
Cluster-1	Gaussian	2.0, 2.0	1.0, 1.0	1, 2	200
Cluster-2	Gaussian	7.0, 2.0	1.0, 1.0	1, 2	200
Cluster-3	Gaussian	12.0, 2.0	1.0, 1.0	1, 2	200

For the second dataset, the distances of the clusters have been kept similar, one of the clusters has been moved to a diagonal position and one more cluster was added in a position blocking a cut between two clusters. This way it should become more difficult for the TUBE clusterer to find axis-parallel cuts to separate them from each other (see Table 6.8, and Figure 6.8).

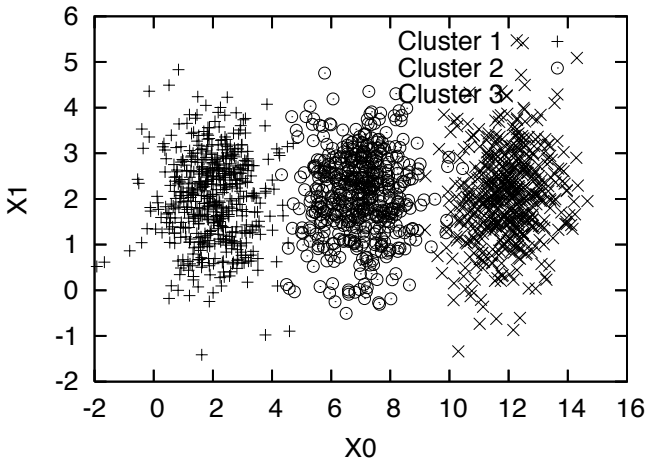

Figure 6.7: Three two-dimensional clusters.

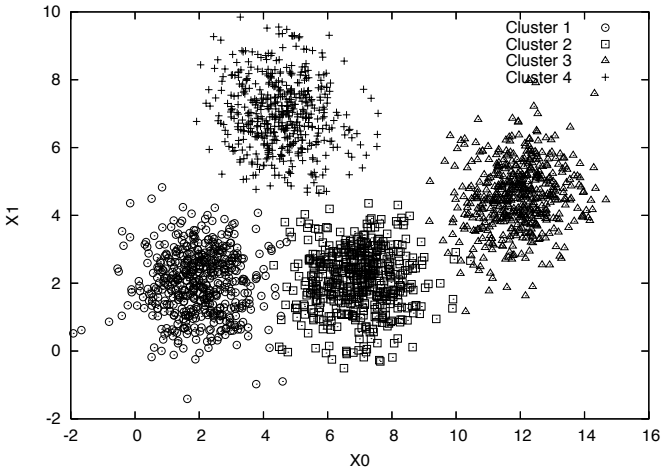

Figure 6.8: Four two-dimensional clusters.

Several datasets with three clusters in a row had been tested with EM-CV until one was found with the clusters just far enough away from each other so that it recognized more than one or two clusters. On this dataset EM-CV found five clusters instead of three. TUBE defines three clusters but has a higher error value. Most errors happen with TUBE first detecting the change in density along the $X1$ axis (the clusters are along the $X0$ axis), which then forms a valley bin and gets assigned to one of the clusters.

In this example the TUBE algorithm found a way to cut around the diagonally positioned clusters so they could be modelled better (with a lower error-rate) than the clusters positioned in a row.

Table 6.8: Example 4: Second dataset - Instructions for the data generator

Example 4 (Second dataset): Four Clusters in Diagonal Position					
	Distribution	Mean	Variance	Attrs	Insts
Cluster-1	Gaussian	2.0, 2.0	1.0, 1.0	1, 2	200
Cluster-2	Gaussian	7.0, 2.0	1.0, 1.0	1, 2	200
Cluster-3	Gaussian	12.0, 4.5	1.0, 1.0	1, 2	200
Cluster-4	Gaussian	4.5, 7.0	1.0, 1.0	1, 2	200

Table 6.9: Example 4: Results overview

Algorithm	Dataset	Num. Clust.	Error Rate	CPU-time
EM-CV	row	5	13.4%	$1m7.210s$
TUBE(100 bins)	row	3	21.2%	$0m1.881$
EM-CV	diagonal	4	0.6%	$0m36.112s$
TUBE(100 bins)	diagonal	4	7.3%	$0m1.980s$

Example 5: Datasets with Oblong-shaped Clusters. Another dataset was generated to have two strongly oblong-shaped clusters. The dataset has two attributes only. See Table 6.10 for the specifications for the generator.

Table 6.10: Example 5: Instructions for the data generator

Example 5: Two Oblong-shaped Clusters				
	Distribution	$Min_x, Max_x, Min_y, Max_y$	Attrs	Insts
Cluster-1	uniform random	2.0, 8.0, 2.0, 3.0	1, 2	500
Cluster-2	uniform random	2.0, 3.0, 8.0, 14.0	1, 2	500

The result with EM-CV finding seven clusters is not surprising. It positioned several spherical shaped clusters along the oblong-shaped clusters. For the TUBE clusterer the generated shapes of clusters are ideal and it finds the two clusters with 0.0% error. The runtime of TUBE and EM-CV are almost the same on this dataset with only few dimensions, only two clusters and relatively few instances.

Example 6: Dataset with Non-Convex Cluster. The dataset for this example was generated with two attributes and one U-shaped cluster. See Table 6.12 for the specifications for the generator and Figure 6.9 for a two-dimensional plot. (The WEKA Subspace clusterer cannot generate U-shaped clusters. To build the U-shaped cluster three oblong-shaped clusters were joined to one cluster by applying a WEKA filter tool.)

The EM-CV algorithm yields similar results as before (Example 5) and

Table 6.11: Example 5: Results overview

Algorithm	Num. Clusters Found	Error Rate	CPU-time
EM-CV	7	53.4%	$0m0.249s$
TUBE(100 bins)	2	0.0%	$0m0.243s$

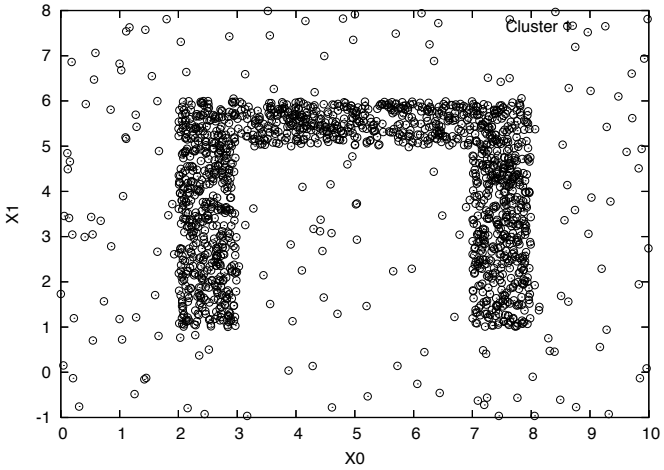

Figure 6.9: Non-convex shaped cluster.

generates several clusters along the shape of the dense areas in the dataset. The TUBE clusterer finds only one cluster, as expected, but adds the surrounding noise data also to the cluster. 12.5% of the instances are noise instances, therefore the result has 12.5% incorrectly clustered instances.

Test on UCI Datasets

For this test the clustering methods TUBE clusterer and EM are used as classifiers to predict the class value. The results in Table 6.14 are here the accuracy and not the error rate (accuracy = 100 - error rate). As test data some of the UCI datasets [6] with nominal class values were selected. The results were evaluated using 10fold-cross-validation, not WEKA's standard classes-to-clusters evaluation as used above. The TUBE clusterer is only designed for numeric attributes so datasets with many or only numeric data were chosen. The following datasets were too large to be evaluated with the EM-based methods in a reasonable testing-time: letter, mfeat-morphol., mfea-zernike, optdigit, pendigit, spambase, waveform-5000 and one test for vowel.

Tube was applied twice, firstly with the evaluation method which assigns one class to only one cluster (column TUBE in Table 6.14) and secondly with an evaluation which allows several clusters for one class, assigning each cluster its training majority class (TU-X). EM was tested three times, searching the

Table 6.12: Example 6: Instructions for the data generator

Example 6: Non-Convex U-shaped Cluster			
	Distribution	$Min_x, Max_x, Min_y, Max_y$	Attributes
Cluster-1	uniform random	$2.0, 3.0, 1.0, 6.0$	$1, 2$
		$3.0, 8.0, 5.0, 6.0$	
		$7.0, 8.0, 1.0, 5.0$	
Noise	uniform random	$0.0, 10.0, -1.0, 8.0$	$1, 2$

Table 6.13: Example 6: Results overview

Algorithm	Num. Clusters Found	Error Rate	CPU-time
EM-CV	15	84.1%	$7m36.229$
TUBE(100 bins)	1	12.5%	$0m1.572s$

number of clusters using cross-validation and the one cluster per class evaluation (EM-CV), setting the number of clusters to the number found by TUBE and the same one cluster per class evaluation (EM-TU), and the majority class evaluation using cross-validation-based model selection (EM-X). The number of classes is listed in column 'Clas' and the number of clusters found by TUBE and EM-CV in columns 'T-c' and 'E-c'.

EM-CV again shows the tendency to generate more clusters than the TUBE clusterer and the accuracy in column EM-CV is often much lower than that of TUBE when the number of clusters (E-c) found is much higher than the real number of classes (Clas).

The results for TUBE are mostly the same with either evaluation method. Classification with EM improves with either method—setting the number of clusters to the number found by TUBE (EM-TU) or using the majority class for the clusters (EM-X). It improves especially when the number of clusters found by EM was higher than the number of classes. In a few cases, TUBE does not find enough clusters (e.g. for the wine data).

Overall, the accuracy is, with some exceptions (breast-w, iris, etc.), far too low, even below fifty percent, which shows that none of the methods are able to build a good enough classification model on many of these datasets. It appears that most areas with a single class are not clustered in a way to form one or several areas of high density surrounded by areas of lower density. EM with the conditional independence assumption and using Gaussian distributions, as implemented in WEKA, finds ball-shaped clusters even when an ellipsoid shape seems to fit better. Along a tube-like shape it defines several clusters and TUBE does define one cluster for such a high density area only.

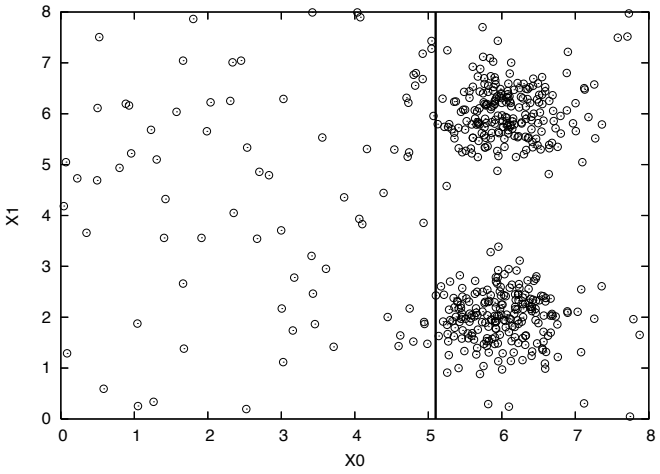

Figure 6.10: The first cut does not define the clusters yet.

TUBE Clusterer: How Many Bins are Needed to Find All Clusters

TUBE binning does not use a fixed bin width but adapts the bin width to the distribution. The TUBE clusterer still needs a certain number of cuts to cut out the areas of higher density. Consider the scatter plot of a two-dimensional dataset with two clusters on one side, which is shown in Figure 6.10. The line shows the first cut chosen. After this cut only one cluster is found. Further binning would cut the two clusters apart.

A fixed number of bins was used for the tests in the above examples. The following test performs the same clustering 99 times, starting with two bins and incrementing the number of bins by one in each iteration. The results in Table 6.15 show the number of bins at which the number of clusters increases.

Summary of the Results

It was shown that the TUBE clustering method can find clusters in multi-dimensional spaces and has the ability to ignore irrelevant attributes. Additionally, the TUBE clusterer also handles clusters in subspaces very well. The generated subspace clusters vary in the set of attributes. The TUBE clusterer's runtime scales very well to larger datasets. It can find clusters of complex shapes and also non-convex high density areas. The precision of the model used to capture these shapes and with that the error rate, depends on the shape of the cluster. The better the cluster fits into a hyper-rectangular box, the better is the precison of the fit.

The examples with generated data and the comparison on the UCI datasets show that EM-CV tends to generate more clusters than the TUBE clusterer.

This can be explained with the nature of the EM algorithm, which best fits convex spherical shapes and fills dense structures or complex oblong shapes with several of these clusters.

6.1.5 Summary for Application Clustering

In this section the applicability of the TUBE clusterer was explored. It was applied to several generated datasets and also to UCI datasets and its error rate and runtime compared with EM-CV, another clustering algorithm which can determine the number of clusters itself. The examples showed that the TUBE clusterer often defines useful clusters and is computationally efficient.

It fulfils several positive characteristics summarised according to Han's and Kamber's list of requirements [24] for clustering algorithms. The above evaluation provide examples of the TUBE clusterer's scalability for large high-dimensional datasets. The test with a non-convex cluster shows that the TUBE clusterer can find clusters of arbitrary shapes.

In the above examples, no parameter had to be chosen according to the domain of the data. The nature of the clusterer allows it to deal well with noisy data and outliers and the representation of the clusters are in interpretable form. The test showed that it can deal well with high-dimensional data because it is scalable to larger datasets and because it can find clusters in subspaces with varying sets of relevant attributes for each cluster in the same dataset.

The number of clusters TUBE finds depends on the number of bins into which the Multi-TUBE binning splits the data and remains constant after a value like 50 or 100 on the datasets considered. On some datasets, cross-validation-based selection of the number of bins in TUBE does not yield the optimum number of clusters.

For future work, several variants of clusterers based on the TUBE binning method could be designed, fitting new applications not yet worked on in this thesis. For a clustering algorithm there is no absolute measureable validation criterion like e.g. the accuracy in classification tasks. It can be said that the usefulness of applying the TUBE clusterer or any other clustering algorithm will always strongly depend on the application. In this thesis the TUBE clusterer was also applied to multiple-instance classification where the evaluation is more objective. This application is explained in the next section (Section 6.2).

Table 6.14: UCI datasets: Results overview with number of classes (Clas), number of clusters found equal to by TUBE (T-c), achieved accuracy using TUBE's clusters for classification (TUBE), and with clusters assigned to classes according to the majority class (TU-X), number of clusters found by EM (E-c), achieved accuracy using EM's clusters for classification (EM-CV);EM with fixed number of clusters to the number of clusters found by TUBE (EM-TU), and with clusters assigned to classes according to the majority class (EM-X).

Dataset	Clas	T-c	TUBE	TU-X	E-c	EM-CV	EM-TU	EM-X
arrhythmia	16	1	47.8	47.8	2	39.8	54.2	56.9
autos	6	3	34.1	33.6	6	39.5	37.6	49.8
balance-scale	3	10	27.8	67.2	3	45.8	23.2	62.4
breast-w	2	2	55.0	65.7	9	56.1	94.0	95.0
cylinder-bands	2	2	55.7	55.7	7	24.6	53.9	64.6
diabetes	2	1	64.7	64.7	7	38.4	65.1	66.8
ecoli	8	2	51.5	51.5	3	77.1	62.2	76.8
glass	7	1	33.6	33.6	7	43.0	35.0	49.1
hayes-roth_test	4	1	50.0	50.0	1	50.0	50.0	50.0
hayes-roth_train	4	7	31.8	42.4	1	38.6	30.3	38.6
heart-statlog	2	1	54.4	54.4	7	43.0	55.6	77.4
ionosphere	2	1	64.1	64.1	10	38.5	64.1	92.0
iris	3	2	53.3	54.7	5	58.7	66.7	90.7
letter	26	2	4.9	4.9	—	—	—	—
liver-disorders	2	1	53.0	58.0	3	43.5	54.5	56.0
mfeat-morphol.	10	4	28.7	28.7	—	—	—	—
mfeat-zernike	10	1	9.7	9.7	—	—	—	—
optdigits	10	3	10.0	10.0	—	—	—	—
page-blocks	5	3	85.6	89.8	6	30.3	89.8	91.3
pendigits	10	3	85.6	85.6	—	—	—	—
segment	7	5	47.2	47.4	13	49.7	69.4	60.3
sonar	2	1	31.7	31.7	8	30.8	51.4	62.5
spambase	2	1	60.0	60.0	—	—	—	—
spectf_test	2	1	64.7	64.7	6	37.5	79.6	79.6
spectf_train	2	1	28.8	28.8	3	67.5	50.0	67.5
spectrometer	4	1	38.4	38.4	11	11.7	42.3	53.1
vehicle	4	2	29.2	29.2	25	22.5	37.0	56.0
vowel	11	1	9.7	9.7	22	—	9.1	19.6
waveform-5000	3	1	32.8	32.8	—	—	—	—
wine	3	1	30.3	30.3	4	75.8	37.1	92.7

Table 6.15: UCI datasets: Number of clusters found

Example Dataset	clusters	2	3	4	5	6	7	8
Ex 1: 15D, 3 clusters	3	16	**29**	-	-	-	-	-
Ex 2: 4D, 3 clusters	3	16	**29**	-	-	-	-	-
Ex 3: 4D, 8 clusters	7	5	13	15	22	32	**40**	-
Ex 4: 2D, 4 in a row	4	10	**12**	-	-	-	-	-
Ex 4: 2D, 4 diagonal	4	9	11	**20**	-	-	-	-
Ex 5: 2D, 2 square clusters	2	**4**			-	-	-	-

6.2 Application: Multiple-Instance Learning

Multiple-instance learning is a new learning paradigm, which has been developed by Dietterich et al. in support of the complex problem of drug activity prediction [14]. The complexity of the problem necessitated a new data representation, which is the *multiple-instance format*. First developed for the prediction of drug activity this new data format is now mostly used for image classification.

In the multiple-instance format an object is not represented by just one instance, but by a *bag of instances*. Each bag is given a class value but the class values of the individual instances in the bag are not known. Like most work on multiple-instance learning, this thesis only considers binary problems where the datasets contain positive and negative objects.

It is assumed that each positive bag contains one or more positive instances that belong to a *positive concept*. Dietterich et al. consider the positive concept to be a contiguous area in multidimensional space in which the positive instances are located. He uses a single multidimensional rectangle to prescribe the positive concept area. In contrast, Maron and Lozano-Perez [44] developed a framework where the difference of the density of positive instances and the density of negative instances is taken to find the positive concept area. In his thesis [43] Maron also allows the option of a concept space that consists of several positive concept areas rather than just one. See Section 6.2.3 for further discussion of different models on how the positive concept area is formed in multiple-instance learning.

In this thesis the multidimensional TUBE binning algorithm, Multi-TUBE is applied to multiple-instance learning in two different ways. In Section 6.2.6 the first application is introduced, which is entirely based on the TUBE clustering algorithm. The second application, an improvement of the time-efficiency of Maron and Lozano-Perez's Diverse Density framework, is explained in Section 6.2.7. This application again uses the TUBE clusterer. For both applications an empirical evaluation is performed and its results are presented.

This section is organized as follows. Section 6.2.1 gives an overview of multiple-instance learning and its principles. Section 6.2.2 introduces well-known existing multiple-instance learning methods and Section 6.2.3 discusses various possibilities to define the concept area. Section 6.2.4 explains the way TUBE clustering is used to define the concept areas. Section 6.2.5 gives an overview of the datasets used for evaluation of both approaches considered: the

TUBE multiple-instance classifier (TUBE-MIC) and the improved version of the Diverse Density algorithm. This algorithms are discussed in Section 6.2.6 and Section 6.2.7 as was mentioned above. In Section 6.2.8 TUBE-based data exploration is performed to elucidate some of the experimental results obtained. Section 6.2.9 summarises the results of the application of Multi-TUBE in multiple-instance learning.

6.2.1 Multiple-Instance Learning

Drug molecules are active when they bind to larger molecules (e.g. proteins) well. How strongly they bind, depends on the shape of their binding site. The molecules of those substances have flexible structures and can form several different three-dimensional conformations, which occur in parallel in the substance. Each conformation has its own shape of binding site. Only one of the conformations needs to bind to make the substance an active one, but often it is not known which one is the one that binds. Each drug is represented by a bag of instances. To facilitate machine learning, each of the instances in the bag represents one of the conformations of the drug molecule. The attributes describing a conformation represent measurements made on the molecule. If the substance was judged to be active, the whole bag containing all conformations of this molecule is labelled as active.

The standard way of representing data in a dataset is that one vector of features represents one instance in the dataset. In the multiple-instance data format one instance is represented by a varying number of feature vectors, called a bag of instances. Each of the instances in a bag has the same number of features. The number of instances in each bag normally varies between the bags in a dataset. Each bag is assigned a class value but the class values of individual instances in the bag are not known.

Note that multiple-instance learning is a different method than learning from *multi-represented* objects [33]. Multi-represented objects are represented by several instances in different feature spaces. In multiple-instance learning an object is also represented by several instances but all of the instances are from the same feature space. Multiple-instance learning sometimes is also called *multi-instance learning*.

6.2.2 Existing Multiple-Instance Learning Methods

The first methods developed for multiple-instance learning were Dietterich et al.'s [14] method of axis-parallel rectangles (APR), Maron and Lozano-Perez's [44] Diverse Density method and an adaptation of the nearest neighbour method to multiple-instance learning by Wang and Zucker [65]. Subsequently other standard machine learning methods were customized for the use in multiple-instance problems, e.g. support vector machines by Andrews et al. [4] and decision trees by Zucker and Chevaleyre [11]. Frank and Xu [22] considered simple methods of transforming a multiple-instance dataset into a normal propositional dataset. After the transformation of the dataset standard machine learning techniques can be applied without any further adaptation. Multiple-instance learning methods with special relevance to the work in this thesis are discussed in more detail in the following.

Dietterich et al.'s Axis Parallel Rectangle method

Dietterich et al. invented the multiple-instances data representation and also introduced the musk datasets on which they tested their algorithms. In the musk datasets each bag contains information about one drug molecule. Each instance in the bag has attribute values representing measurementd pertaining to one conformation of the drug molecule. A bag can be either of class *active* (positive) or of class *inactive* (negative).

Dietterich et al.'s learning algorithm searches for an estimate $\hat{f}$ of the unknown function f, which is the function that labels the bags correctly. Let M be the set of all possible feature vectors of length n, and $m_{i,j} \in M$ be one of the feature vectors of bag m_i. Dietterich et al. say that in the multiple-instance case it is not a single-instance function that is the ultimate goal, but a multi-instance function $f(m_i)$ where $m_{i,1}, m_{i,2}, ..., m_{i,\nu_i}$ are the ν_i ambiguous variants of an (unknown) feature vector m_i. They define function f based on a single-instance function g.

$$f(m_i) = \begin{cases} 1 & if\ \exists j \quad g(m_{i,j}) = 1 \\ 0 & otherwise \end{cases} \tag{6.1}$$

where $f(m_i) = 0$ for inactive bags and $f(m_i) = 1$ for active bags. This models the assumption that in a positive bag at least one ambiguous variant must be positive (active).

The goal of the multiple-instance algorithm is to construct an approxima-

tion $\hat{g}$ of the internal function or in other words to build a model of the positive concept that classifies at least one instance from each positive bag as positive and all instances from the negative bags as negative. In Dietterich et al.'s work the model of the positive concept is one axis-parallel hyper-rectangle (APR). Three different algorithm variants are described:

1. The 'standard' algorithm generates the smallest APR that contains all positive instances. This algorithm effectively ignores the multiple-instance nature of the problem.

2. The 'outside-in' algorithm takes the APR of the 'standard' algorithm and shrinks it to exclude all false-positive instances. The shrinking process considers the multiple-instance nature of the problem.

3. The 'inside-out' algorithm starts with the APR representing a single positive instance and grows it with the goal to cover at least one instance of all positive bags and no instance of the negative bags. In this way the algorithm also takes the multiple-instance nature of the problem into account.

In [14] Dietterich et al. find that the APR variants that take the multiple-instance nature of the problem into account show better results and that the 'inside-out' algorithm performs best. They evaluated these methods on the musk drug discovery datasets.

TUBE's similarity to the APR methods is that TUBE's binning also results in axis-parallel rectangles. However, TUBE's method of finding the rectangles differs from APR's algorithm in that it can be used to generate a cluster of rectangles as the model for the positive concept area instead of one rectangle alone. The method is explained in detail in Section 6.2.4.

Weidmann et al's Two-Level Classification Approach

Weidmann et al. [67] developed a two-level classification approach assuming that each instance in the bag is potentially contributing to the class label and called it multi-instance learning.

Weidmann at al.'s algorithm designed the two-level classification approach for generalized multi-instance learning. In the generalized scenario it is assumed that the relationship between the class labels of a bag's instances and the bag's label can be more complicated than in the scenario considered by Dietterich et al.

Weidmann et al.'s algorithm implements the two-level classification approach in a heuristic manner. To reach the first level each bag is transformed into a meta-instance, resulting in a new dataset that has as many instances as there are bags in the original dataset. In the second level, the data can thus be classified by any propositional learner and the method does not require learners that are adapted to the multi-instance format. The transformation is done by decomposing the instance space into 'candidate' regions for each concept using a decision tree to split the instance space according to the relative number of positive and negative instances. The regions found become the attributes of the newly formed meta-instances. Attribute values for a meta-instance corresponding to a bag are computed based on the distribution of the bag's instances into these regions. Weidmann et al. compare this method with the Diverse Density algorithm [43] and the MI Support Vector Machine [4] on several artificial datasets representing generalized multiple-instance problems.

Weidmann et al.'s approach is related to the methods introduced in this thesis because it also uses a tree structure to create rectangular regions. However, their method is explicitly focused on the application to generalized multiple-instance problems.

Maron and Lozano-Perez's Diverse Density Algorithm

Maron and Lozano-Perez [43][44] use density measures to model the probability that a bag is positive. The positive concept is viewed as a 'fuzzy' point in the instance space, defined by the location where the instance areas of all positive bags intersect without intersecting any of the negative bags. Using only a single point would mean a rather severe restriction for the model of the positive concept, therefore the method uses probability theory to define a 'fuzzy' area around this point. This prevents overfitting, which would exclude too many bags by classifying them as negative.

To find the positive concept Maron and Lozano-Perez use a new measure called *Diverse Density*. This is defined for each point in the instance space and measures how many different positive bags have instances near that point, and how far the negative instances are from that point. This is defined as a probabilistic measure. The algorithm returns the location of the positive concept point that represents a local maximum of Diverse Density and also a weight vector that emphasizes important attributes. The concept area has a multidimensional ellipsoid form.

Maron derives the probabilistic model of the positive concept in the fol-

lowing way: Assume that the positive concept is a single point x and that B_i^+ is the i-th positive and B_i^- the i-th negative bag. The idea is to find the optimal point for the positive concept by maximising the probability that x is the positive concept given the positive and negative bags.

$$arg\ max_x(P(x = t|B_1^+, ..., B_n^+, B_1^-, ..., B_m^-)) \tag{6.2}$$

To compute this Maron and Lozano-Perez use Bayes' rule and an uninformative prior over all locations and make the assumption that the bags are conditionally independent given the target concept. After using Bayes' rule again this results in a general definition of the maximum Diverse Density:

$$arg\ max_x \prod_i P(x = t|B_i^+) \prod_i P(x = t|B_i^-) \tag{6.3}$$

Using the noisy-or model to define the terms in the products and assuming the jth point in the bag B_i is B_{ij} yields:

$$P(x = t|B_i^+) = P(x = t|B_{i1}^x, .., B_{in}^x) = 1 - \prod_j (1 - P(x = t|B_{ij}^+)) \tag{6.4}$$

The probability for a negative bag $P(x = t|B_i^-)$ is defined as one minus the term on the right-hand side of this equation. The probability $P(x = t|B_{ij})$ that an instance is similar to a certain positive concept is relative to the distance of the point to the positive concept:

$$P(x = t|B_{ij}) = exp(-||B_{ij} - x||^2) \tag{6.5}$$

Note that with the number of positive intersecting bags the quantity in Equation 6.2 grows in an exponential way.

A gradient ascent method is used to find a maximum for Equation 6.2, but this can only be a local maximum. Therefore gradient ascent is repeated several times choosing the instances of the largest positive bag or bags as starting points. This makes the computation very time consuming.

Maron [43] tested the Diverse Density algorithm on the musk datasets and on a generated dataset. He further applied it to image recognition and stock selection, with the aim of showing that the algorithm can deal with a high level of noise. In his thesis, Maron [43] also offers a variant of the method with several positive concepts instead of one. However, this method is even more computationally complex.

The Diverse Density algorithm is one of the standard algorithms that are applied to multiple-instance problems and it often achieves very competitive classification accuracy. Unfortunately the method has unacceptable runtime behaviour especially when applied to larger datasets. Section 6.2.7 explains how the runtime behaviour can be improved using TUBE while maintaining high accuracy.

6.2.3 Models for the Positive Concept

Although MI-Learning is a fairly new domain in machine learning there is already a traditional view of how a positive concept is formed. Weidmann [67] named it the *standard multiple-instance assumption*. This and other models are briefly discussed below.

Standard Multiple-instance Assumption This assumption is that the class value of a bag is determined by the presence or absence of at least one positive instance; the other instances can actually be viewed as negative instances even if they are in a positive bag. Dietterich et al. [14] argued that the negative instances have no impact on the class value and that therefore all negative instances in positive bags can effectively be regarded as noise. Some of the APR methods and the Diverse Density method are explicitly based on this assumption.

Long and Tan [41] give a succinct expression for the standard assumption, with x_i being the k feature vectors in a bag and $f(x_i)$ being the function that determines the class value of an instance (either positive or negative) [1]:

$$(\{x_1, x_2, ..., x_k\}, f(x_1) \vee f(x_2) \vee ... \vee f(x_k))$$

i.e. the bag's class label is the disjunction of its instances' class label.

'Whole Bag'-Theory Weidmann et al. [67] define the 'Whole Bag'-theory which states that an interaction between instances in a bag determines its class label. This means that the negative instances in positive bags can be part of the concept and the data is represented as:

$$(\{x_1, x_2, ..., x_k\}, f(x_1, x_2, .., x_k))$$

[1] Remember: In the case of a standard, single-instance learning, with one instance representing each object, the representation of an instance is $(x, f(x))$.

using a more general function f.

Looking at the drug activity prediction problem and at a drug that is active, then it is likely that only one conformation of the drug molecule is active. But perhaps its negative conformations differ from those of other negative drugs. The physical properties of the molecule enable the drug to transform from its negative, non-active conformation to an active conformation.

The Number of Concept Areas The APR methods look for only one single axis-parallel rectangle to capture the positive concept. In contrast Maron [43] considers multiple-instance problems with one or several concepts. Weidmann et al. [67] generalize further and specify variations of problems with several concept areas so that a positive bag must have instances in each of them. They differentiate between three classes of problems, one which requires only one instance in each positive concept area, a second with a certain sufficient minimum number of instances for each area, and a third one with a minimum and a maximum number of instances in each area.

Shape of the Concept Area A concept can be seen as a point or a cloud of instances in the concept space, or a set of those. The APR methods are based on using a multidimensional rectangle as model for the positive concept. The Diverse Density method is based on a 'soft' probabilistically defined area around a target point. Weidmann et al. use sets of rectangular areas to represent the target concept.

6.2.4 Using TUBE to Define the Positive Concept Area

The technique explained in this section finds the positive concept area for a multiple-instance learning problem using the multidimensional TUBE histogram. The same technique is part of both of the two applications of Multi-TUBE to multiple-instance learning: the TUBE multiple-instance classifier and the improved version of the Diverse Density algorithm, which are discussed in the following sections, Sections 6.2.6 and 6.2.7.

TUBE can define a positive concept area that is more complex than APR by gathering a cluster of multidimensional rectangles. In order to do this, Multi-TUBE generates a multidimensional histogram. Then the TUBE clusterer is used to locate the mode bins in this histogram and gather the neighbouring bins down to the valley bins to form the clusters. Depending on the application a density threshold can be set to exclude bins below a certain density.

The standard histogram would be of no use for this task. The concept area in multiple-instance learning is an area with positive instances from positive bags but very few instances from negative bags. Maron [43] uses the new Diverse Density measure to define the optimal point of the positive concept. With some similarity to Maron's approach, the TUBE difference density technique models the concept area looking for multidimensional rectangles with high difference of densities D_{diff}. With n_i^+ being the number of positive instances in bin_i, and n_i^- being the number of negative instances in bin i, the volume of the bin being v_i, and the total number of instances being N, the difference of densities D_{diff} is defined as follows:

$$D_{diff} = \frac{n_i^+}{(v_i \times N^+)} - \frac{n_i^-}{(v_i \times N^-)} \tag{6.6}$$

To find the rectangles that best represent the concept area the TUBE algorithm 'mixing of binnings' (see Section 5.4.1 and Algorithm 3 in the previous chapter) is used. In the following, this technique is again briefly summarised and it is explained how it is used in the context of multiple-instance learning.

To prepare for the mixing, the instances of all bins are gathered in two new datasets, one for the positive instances and one for the negative instances, with the instances inheriting their class value from the corresponding bags. Two multidimensional histograms are formed with a fixed number of bins (in the experiments in this section, 5 or 10 bins) using the two training datasets. Then the two binnings are combined into one histogram on their joint range.

Finally the TUBE clusterer locates all mode bins and the clusters in the histogram using the Difference Density measure. Using this measurement to define the modes is again specific to the multiple-instance scenario. The TUBE clusterer has been explained in the previous chapter in Section 6.1. The modes in the difference density histogram are the areas where the difference between the densities of the positive and the negative instances is the largest and are therefore good candidates for the concept area.

The mixing of binnings can quickly result in a shredded result range. Therefore, the two subdatasets must be split into a small number of bins (e.g. 5 and 10) for this method. This was done for all experiments in both applications of Multi-TUBE in multiple-instance learning.

Table 6.16: Basic statistics of the datasets used.

Dataset	num-bags	num-inst	min-inst	max-inst	num-attr
eastwest	20	213	4	16	24
elephant	200	1391	2	13	230
fox	200	1320	2	13	230
musk1	92	476	2	40	166
musk2	102	6598	1	1044	166
mutagenesis3atoms	188	1618	5	15	10
mutagenesis3bonds	188	3995	8	40	16
mutagenesis3chains	188	5349	8	52	24
tiger	200	1220	1	13	232
thioredoxin	193	26611	35	189	8
westeast	20	213	4	16	24

6.2.5 Datasets

Table 6.16 shows some statistics summarising the datasets used in the experiments. The number of bags (first column) varies between 20 and 200 and the datasets have from 8 to 232 attributes (last column). The column *num-inst* lists the number of instances over all bags for each dataset. The two columns *min-inst* and *max-inst* give the minimum number and the maximum number of instances found in a single bag of the dataset. The *musk1* dataset shows the largest variation in number of instances per bag here ($min = 1$ and $max = 1044$). Many of the datasets have some bags with very few (≤ 2) instances. More information on these datasets can be found in [19].

6.2.6 Using TUBE Clusters for MI Classification

This section discusses a new multiple-instance classifier, the *TUBE multiple-instance classifier* (*TUBE-MIC*), that is directly based on the binning generated by Multi-TUBE using the process described above in Section 6.2.4. It constructs clusters of multidimensional rectangles as a model for the positive concept area. TUBE-MIC calls the TUBE clusterer to find the clusters. The binning is based on the mixing of binnings method and the method uses the difference density D_{diff} (see Equation 6.6) instead of the density for each bin. Thus, TUBE-MIC finds a positive concept area that is a collection of connected multidimensional rectangles with high difference between positive instance density and negative instance density.

The prediction of the class of a new bag is straightforward in this approach. TUBE-MIC simply checks if one of the instances in the bag falls into the concept area it has identified based on the training data, to predict it as a

Table 6.17: Multiple-instance Diverse Density classification (MIDD) compared with TUBE-MIC with the thresholds set to 90 (TMIC-90), 70 (TMIC-70) and 60 (TMIC-60) respectively (v significant win, * loss against DD).

	MIDD	TMIC-90	TMIC-70	TMIC-60
eastwest	61.50	80.00 v	80.00 v	80.00 v
elephant	80.20	65.15 *	66.90 *	67.55 *
fox	61.05	55.45 *	54.55 *	54.65 *
musk1	86.17	65.49 *	69.54 *	69.32 *
musk2	82.18	63.05 *	63.52 *	63.13 *
mutagenesis3-atoms	72.46	48.49 *	51.68 *	67.70 *
mutagenesis3-bonds	75.25	70.37 *	69.72 *	66.54 *
mutagenesis3-chains	78.33	58.25 *	71.18 *	66.60 *
tiger	64.32	62.15	64.85	65.85
thioredoxin	76.37	80.07 v	79.56 v	78.36 v
westeast	79.78	41.00 *	41.00 *	41.00 *

positive bag. TUBE-MIC uses a density threshold and bins with lower density than the threshold are discarded from the positive concept cluster. A threshold is given in percent format per user parameter, with the highest density in the histogram represented by the value 100 percent.

Evaluation

The method was tested on the multiple-instance datasets in Table 6.16, using ten bins for the mixing of binnings algorithm and several different density thresholds. Table 6.17 shows the accuracy values of the TUBE-MIC classifier that were achieved, compared with the results of the Diverse Density classifier in WEKA [68]. All experiments were performed using 10×10-cross-validation and all comparisons are based on the corrected resampled t-test [47].

The results show that except for the eastwest dataset the result achieved by the Diverse Density classifier could not be improved substantially. In fact in most other cases except tiger and thioredoxin, performance was worse. Thus, the 'soft' ellipsoid concept found by the Diverse Density algorithm appears more appropriate for most of these datasets.

The result on the eastwest data will be explored further in Section 6.2.8 using the histogram representation method bin list. For most datasets the results vary with the different density threshold values. Lowering of the threshold improved the result substantially on the mutagenesis3 datasets. The reason for this is that this adjustment added one or several more significant lower density

bins to the positive concept cluster area.

6.2.7 Improving the Efficiency of the Diverse Density Algorithm

Maron's Diverse Density algorithm (MIDD) uses the Diverse Density measure to find the positive concept area. It is started several times for each instance of the largest positive bag(s), to avoid a local maximum solution. Using instances of a positive bag increases the probability that the process starts from a point where the density of the 'real' positive instances is high and the maximum can be found faster. Nevertheless, the computational requirements of the DD algorithm remain very high.

The TUBE-based improvement to the Diverse Density method discussed in this section has the aim of shortening the search for the global maximum by providing the algorithm with better starting points. The idea is to again use the difference of the densities function of the positive instances and the density of the negative instances to generate a difference density histogram. The modes of the difference density function are areas were the density of the positive instances, which are the instances from positive bags, is high compared to the density of the negative instances (the instances from negative bags) and thus more likely to be areas of 'real' positive instances. Hence they are a good area to harvest starting points for the Diverse Density algorithm

The actual process of generating the density estimator and finding the modes is in most parts the same as in the previous application (Section 6.2.6) and uses the mixing of binnings to build a multidimensional histogram and the TUBE clusterer to cluster the bins. However, in this application only the mode bins of these clusters are utilized and the clusters are irrelevant. Either the centre point of each mode bin is taken as starting point or one of the training instances in this bin is randomly selected for this purpose.

Evaluation

The method was tested on the datasets introduced previously, with several different parameter settings: either using five bins (TUBE-5) or ten bins (TUBE-10) in the histograms generated for the mixing of binnings. TUBE-5 and TUBE-10 both use the centre of the mode bins, that have been found as starting points for the Diverse Density algorithm. The third variation is again based on using ten bins but instead of using the centre points, picks one randomly

Table 6.18: Multiple-instance Diverse Density classification (DD); accuracy compared with TUBE-augmented method with five bins (TUBE-5), ten bins (TUBE-10) and ten bins with the starting instances selected randomly (TUBE-10-rand) (v significant win, * loss against DD).

	MIDD	TUBE-5	TUBE-10	TUBE-10-rand
eastwest	61.50	66.00	67.00	69.50
elephant	80.20	77.45	78.90	79.00
fox	61.05	59.75	58.60	58.00
musk1	86.17	48.89 *	54.47 *	67.16 *
musk2	82.18	61.73 *	61.73 *	82.67
mutagenesis3-atoms	72.46	72.25	72.72	72.93
mutagenesis3-bonds	75.25	75.41	75.17	74.85
mutagenesis3-chains	78.33	76.31	78.76	76.00
tiger	73.20	50.60 *	69.35	63.45
thioredoxin	90.06	84.43 *	84.54 *	84.38 *
westeast	37.50	28.50	27.00	31.00

selected training instance from each mode bin as the starting points (TUBE-10-rand). All experiments were performed using 10×10-cross-validation and all comparisons are based on the corrected resampled t-test [47].

Table 6.18 shows the classification results of the Diverse Density (MIDD) algorithm compared with the accuracy achieved with the Diverse Density classifier augmented with TUBE-based starting point selection. For all datasets but musk1, musk2 and thioredoxin the TUBE method achieves similar results as the plain Diverse Density classification. The last column shows that the use of a randomly picked training instance enables the TUBE version to perform equivalently to DD on the musk2 dataset but not the musk1 dataset. The musk datasets are two very similar, high-dimensional datasets both containing drug prediction data. Section 6.2.8 explores the musk datasets and analyzes their structure with the aim of explaining why the method does not work for the musk1 dataset.

Picking an instance randomly from the mode bin yielded a substantial improvement on the musk2 dataset. This could be because in the multidimensional bin, most instances lie closer to the borders than to the centre point of the bin.

Table 6.19 lists the CPU training times required by the experiments and shows that the TUBE-augmented method is in all cases significantly faster, yielding a practical algorithm with good performance in many cases.

Table 6.19: CPU time comparison: Diverse Density method (DD), TUBE augmented method with five bins (TUBE-5), ten bins (TUBE-10) and ten bins with the starting instances selected randomly (TUBE-10-rand) (v significantly faster, * significantly slower than DD).

	MIDD	TUBE-5	TUBE-10	TUBE-10-rand
eastwest	3.56	0.21 v	0.24 v	0.25 v
elephant	438.58	48.74 v	33.75 v	35.48 v
fox	229.36	35.21 v	33.85 v	33.27 v
musk1	29.65	0.55 v	1.32 v	3.05 v
musk2	2639.09	5.21 v	6.66 v	52.92 v
mutagenesis3-atoms	19.23	1.51 v	1.61 v	0.87 v
mutagenesis3-bonds	111.00	3.10 v	3.81 v	2.15 v
mutagenesis3-chains	445.14	4.86 v	7.04 v	3.56 v
tiger	166.97	5.59 v	23.67 v	19.19 v
thioredoxin	1332.52	15.21 v	5.62 v	6.17 v
westeast	1.93	0.38 v	0.17 v	0.18 v

6.2.8 Elucidating the Results Using Data Exploration

In this section, the eastwest dataset is explored in more detail in an attempt to explain the good results obtained with the TUBE multiple-instance classifier (TUBE-MIC) on this dataset. In addition, the musk1 and musk2 datasets are explored to analyze Diverse Density multiple-instance classification with TUBE. As mentioned above, for the musk2 dataset, the accuracy achieved using TUBE's starting points for the Diverse Density classifier was comparable to the results achieved with Diverse Density classification by itself. This goal could not be achieved for the musk1 dataset.

The histograms generated as part of the learning process are represented using bin lists. This technique is explained in detail in the previous chapter, in Section 5.5. The binning shown in the bin lists is the same as that used in the classification tasks (Section 6.2.6), although only when the algorithm is applied to the whole dataset and not within the cross-validation. As a result of the mixing of binnings the bins are ordered according to their difference density (D_{diff}).

In the following sections the structure found is briefly explained and the bin lists are given. Whenever similar bins are repeated in a bin list they are substituted by a row with "...". The bin lists with all non-empty bins listed are given in Appendix A.

The *eastwest* Dataset

The mixing of binnings applied to the eastwest dataset resulted in 509 bins. All areas are very mixed with positive and negative instances. 4 of the 26 attributes in this data have been cut. Nine clusters were found. The information for the nine corresponding mode bins is shown in Figure 6.11. Five of the clusters' mode bins contain only positive instances, many of the bins only negative instances. The last cluster has a mode with zero instances in it, surrounded by bins of mostly negative instances. Mode bins with zero positive instances are not used for the harvesting of starting points. Only two of the nine clusters are shown in Figure 6.11. (For the more complete bin list see Appendix A.)

```
Cluster Bin List: 0
Mode Bin: 0

0  :AbbbbbbbbbbB Dns:[XXXXXXXXXX] Ins:[X.........] Vol:[X.........] 3.76%
1  :Abbbb......B Dns:[XXXX......] Ins:[X.........] Vol:[X.........] 2.82%
2  :-.........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
...
...
75 :-.........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
76 :Ba.........A Dns:[X.........] Ins:[X.........] Vol:[X.........] 0.94%
77 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
...
85 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%

Cluster Bin List: 1
Mode bin: 0

0  :AbbbbbbbbbbB Dns:[XXXXXXXXXX] Ins:[X.........] Vol:[X.........] 3.76%
1  :Abbbb......B Dns:[XXXX......] Ins:[X.........] Vol:[X.........] 2.82%
2  :-.........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
...
...
75 :-.........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
76 :Ba.........A Dns:[X.........] Ins:[X.........] Vol:[X.........] 0.94%
77 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
...
84 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
85 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%

...
(For Cluster Bin Lists 3 - 8 see appendix.)
```

Figure 6.11: Bin list of the eastwest datasets. Nine clusters found.

The *musk1* Dataset

The bin list of the multidimensional histogram built for the musk1 dataset contains 584 bins. All areas are very mixed with positive and negative instances. 11 of the 168 attributes have been cut. Only one cluster was found. The bin list of this cluster is shown in Figure 6.12. This means when applied to Diverse Density it will use one instance only as starting point. The mode bin is a very small bin with a comparably large number of instances (15.13 percent). The density of negative instances in the mode bin is between 30 and 40 percent. (For the more complete bin list see Appendix A.)

```
Cluster Bin List: 0
Mode bin: 0
0  :0aaaabbbbbbB Dns:[XXXXXXXXXX] Ins:[XX........] Vol:[<1E-17....] 15.13%
1  :0a.........B Dns:[X.........] Ins:[X.........] Vol:[<1E-15....] 3.15%
2  :Ab.........B Dns:[<0.1......] Ins:[X.........] Vol:[<1E-15....] 0.21%
...
23 :0a.........B Dns:[<1E-12....] Ins:[X.........] Vol:[<0.0001...] 0.42%
24 :-..........- Dns:[          ] Ins:[          ] Vol:[<1E-17....] 0.0%
...
...
522:-..........- Dns:[          ] Ins:[          ] Vol:[XXXXXXXXX.] 0.0%
523:Ba.........A Dns:[<1E-16....] Ins:[X.........] Vol:[<0.1......] 0.21%
524:Ba.........A Dns:[<1E-15....] Ins:[X.........] Vol:[<0.1......] 0.21%
...
581:0b.........A Dns:[X.........] Ins:[X.........] Vol:[<1E-15....] 7.35%
582:Ba.........A Dns:[<0.1......] Ins:[X.........] Vol:[<1E-16....] 0.21%
583:Ba.........A Dns:[X.........] Ins:[X.........] Vol:[<1E-16....] 0.63%
```

Figure 6.12: Bin list of the musk1 datasets. Only one cluster found.

The *musk2* Dataset

The mixing of binnings resulted in 722 bins for the musk2 dataset. As in the case of the musk1 datasets areas are very mixed with positive and negative instances. 9 of the 168 attributes have been cut. The TUBE clusterer found two clusters. Thus it provides two starting points when applied in conjunction with the Diverse Density algorithm. The mode of the first cluster is a small area with only a small percentage of training instances but is quite pure with respect to the positive instance class. The bin lists of the two clusters are shown in Figure 6.13. (For the more complete bin lists see Appendix A.)

```
Cluster Bin List: 0
Mode Bin: 0

0  :0abbbbbbbbbB Dns:[XXXX......] Ins:[X.........] Vol:[<1E-15....] 1.85%
1  :0aaaaabbbbbB Dns:[XXXXXXXXXX] Ins:[XX........] Vol:[<1E-14....] 19.38%
2  :Abbb.......B Dns:[X.........] Ins:[<0.1......] Vol:[<1E-16....] 0.06%
3  :0ab........B Dns:[XX........] Ins:[X.........] Vol:[<1E-15....] 0.41%
...
26 :0a.........B Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.09%
27 :-..........- Dns:[          ] Ins:[          ] Vol:[<1E-16....] 0.0%
...
...
269:-..........- Dns:[          ] Ins:[          ] Vol:[XXXXXXXXX.] 0.0%
270:Ba.........A Dns:[<1E-15....] Ins:[<0.1......] Vol:[X.........] 0.08%
...
372:Ba.........A Dns:[X.........] Ins:[<0.1......] Vol:[<1E-14....] 0.02%
373:0b.........A Dns:[X.........] Ins:[X.........] Vol:[<1E-15....] 0.11%
374:0b.........A Dns:[X.........] Ins:[X.........] Vol:[<1E-13....] 4.88%
375:0b.........A Dns:[X.........] Ins:[X.........] Vol:[<1E-14....] 0.17%

Cluster Bin List: 1
Mode Bin: 0

0  :0a.........B Dns:[<0.01.....] Ins:[X.........] Vol:[<1E-12....] 0.32%
1  :Ab.........B Dns:[<0.001....] Ins:[X.........] Vol:[<1E-11....] 0.17%
2  :0a.........B Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-10....] 0.18%
...
7  :Ab.........B Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.09%
8  :0a.........B Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.09%
9  :-..........- Dns:[          ] Ins:[          ] Vol:[<1E-16....] 0.0%
...
...
245:-..........- Dns:[          ] Ins:[          ] Vol:[XXXXXXXXX.] 0.0%
246:Ba.........A Dns:[<1E-15....] Ins:[<0.1......] Vol:[X.........] 0.08%
247:Ba.........A Dns:[<1E-14....] Ins:[X.........] Vol:[X.........] 0.14%
...
345:Ba.........A Dns:[<0.01.....] Ins:[<0.1......] Vol:[<1E-12....] 0.03%
346:Ba.........A Dns:[<0.1......] Ins:[<0.1......] Vol:[<1E-14....] 0.03%
347:0b.........A Dns:[X.........] Ins:[X.........] Vol:[<1E-15....] 0.11%
```

Figure 6.13: Bin list of the musk2 datasets. Two clusters found.

Evaluation

The eastwest dataset histogram was examined to investigate the good performance of TUBE-MIC classification on this data (see Section 6.2.6). Considering the bin list, the positive area appears to be clearly defined: many of the mode bins and some of their neighbouring bins contain positive instances only. This makes it easy for the TUBE multiple-instance classifier to find the positive concept area and results in the high accuracy observed.

The musk1 and musk2 datasets are very similar datasets both containing information about drug activity prediction but still show quite different results when processed with Diverse Density classification using TUBE (see also Section 6.2.7). The musk2 dataset is larger than musk1. Because it has more instances to train a nonparametric density estimator, the resulting histogram can better model the underlying density function and can thus provide DD classification with better starting points. Multi-TUBE binning finds more structure in musk2 than in musk1: two clusters instead of one. Moreover the mode bin of musk2's first cluster has a relatively small density of negative instances, which makes it a better source of starting points for DD classification.

6.2.9 Summary Application Multiple-Instance Learning

In this section TUBE clustering was applied in two ways, in both cases in the area of multiple-instance learning. One new multiple-instance classification algorithm was developed, and one existing algorithm augmented with TUBE primarily to improve its runtime. Both applications employ the same techniques, presented earlier in this thesis: Multi-TUBE's multidimensional histograms, the mixing of binnings and the TUBE clusterer. A novel aspect of these applications is that the positive concept area is identified based on the bins of the histogram generated using the difference density D_{diff} instead of the density.

The first application, the TUBE multiple-instance classifier (TUBE-MIC), predicts the class value of new bags directly if any of the instances in the bag fall into the positive concept area. TUBE-MIC takes a cluster as model that consists of several potentially neighbouring bins surrounding a mode bin. The borders of the cluster are confined by a density threshold that is given per user parameter. Several multiple-instance datasets were tested with this method and the performance compared with that of the Diverse Density classifier. One observation was that the best density threshold to use appears to differ between datasets, which makes adjustment to each dataset necessary. Except for the eastwest dataset, and perhaps thioredoxin and tiger, the classification results were disappointing compared to the (albeit much slower) Diverse Density method. Further analysis of the bins that were generated shows that the eastwest dataset has areas containing exclusively positive instances, which makes it easy for TUBE-MIC to define a good model.

In the second application, the search for the concept area point performed

by the Diverse Density classifier [43] was supported by supplying the algorithm with a small set of high-quality starting points using TUBE. The proposed method takes the starting points from the mode bins of Multi-TUBE's histogram estimator. As mentioned above, the histogram uses the difference density, so that the mode bins define a range in which the density of the positive instances is high compared to that of the negative instances. This improved the runtime of the Diverse Density classifier significantly. The accuracy was comparable in most cases, with only the musk1 and thioredoxin datasets exhibiting substantially worse accuracy. Further exploration of the musk1 dataset using the bin list representation technique, showed that the multidimensional histogram only contained bins mixing positive and negative instances, which makes it difficult to identify good mode bins for the selection of starting points.

Both application ignored the standard multiple-instance assumption —that the positive concept area should have at least one instance of each positive bag in it—in the process of identifying relevant rectangular regions using TUBE. For the first application taking it into account did not show any improvement in the results and in the second application, improving the DD algorithm, the number of starting points was already sufficiently reduced for each of the datasets.

6.3 Summary

In this chapter the new multivariate tree-based density estimation methods proposed in this thesis have been applied to clustering and to multiple-instance learning. The chapter contains two corresponding sections. The first section discusses the newly developed TUBE clustering algorithm. In the second section, the TUBE clusterer is first used directly for multiple-instance classification and secondly in support of an existing algorithm for multiple-instance classification.

Section 6.1 explains the newly developed TUBE clusterer. It generates multidimensional clusters based on the Multi-TUBE binning. The TUBE clusterer exhibits several positive attributes such as the ability to find subspace clusters, good scalability and it can also handle data with noise well. The clusters found are not restricted to be simple geometric shapes but can be of complex shape.

The second application, covered in Section 6.2, concerns the field of multiple-instance learning. Multiple-instance learning has a special data format, with each example represented by a bag of instances. A class value is assigned to each bag but the actual class of each single instance is not known. The standard multiple-instance assumption is that each negative bag contains only negative instances but that each positive bag contains one or more positive instances. Based on this assumption each multiple-instance problem has a positive concept area where the positive instances are found.

Using Multi-TUBE as the underlying algorithm, a model for classifying future bags is identified by finding areas with 'real' positives instances. The precision of this method depends on the level of false negative instances in the negative bags. The positive areas are very fragile with only a few instances being 'real' positives and could easily be covered up by a noisy false positive bag. The assumption is that the negative instances in the positive bags are not different to the negative instances in the negative bags and fall into the same negative concept areas.

The TUBE multiple-instance classifier (TUBE-MIC) defines the concept area by generating clusters of bins using the difference density. TUBE-MIC predicts class values directly: if one instance in a test bag is in the positive concept cluster, the bag is declared positive. In this way it can model complex non-convex concept areas. However, classification with TUBE-MIC was disappointing compared with the accuracy results of the Diverse Density classifier. The reason for the unsatisfying results could be due to overfitting of the

model to the concept area. Future work should provide 'soft' cluster borders. Further development of TUBE-MIC could also take into account the fact that a positive concept area should contain a positive instance of each positive bag from the training data.

In the second application discussed in Section 6.2, a TUBE-based method was used in support of the Diverse Density classifier. This classifier is based on maximising the Diverse Density measure and looks for areas with high density of positive instances and low density of negative instances. Diverse Density repeatedly applies an optimization algorithm (gradient ascent) with several starting points. The TUBE-based improvement takes instances from the mode bins of the multivariate histogram to provide high-quality starting points and so can quicken the process of generating a Diverse Density classifier in almost all cases without significant loss of accuracy.

The next chapter summarizes the work done in this thesis and lists the conclusions which were drawn from it.

Chapter 7

Conclusions

In this thesis new unsupervised, tree-based algorithms for the discretization of univariate and multivariate ranges were developed and used to construct density estimators. These were applied in several machine learning tasks: discretization for density estimation itself, naive Bayes classification, clustering and multiple-instance classification. The research questions are:

- Can greedy algorithms for tree-based density estimation represent the significant structure of the distribution function well, by adapting to all changes in the density function, and also abrupt changes and areas of (effectively) zero density? For instance, can tree-based density estimation be used to find clusters in multidimensional data?

- Can the algorithms employ cross-validation to determine an appropriate number of bins based on the input data alone?

- Can the induced nonparametric density estimators support density estimation tasks in machine learning applications such as single-instance and multiple-instance classification, so that the performance in the target application is improved when the new estimator is used to augment an existing approach or replace a standard histogram estimator?

- Can tree-based density estimators generate density models based on computational requirements that render them useful in practical applications?

In the next section the work done in this thesis is summarised and answers to these questions are provided. Section 7.2 highlights the main contributions.

Section 7.4 lists the conclusions that can be drawn considering these questions. Section 7.5 proposes further possible work based upon the developed algorithms.

7.1 Summary

Greedy tree-based approaches are the basis for the algorithms developed and evaluated in this thesis. It was shown that they can support discretization, clustering and can make classification algorithms more efficient. The greedy tree-based approach was chosen to enable an efficient search for bin thresholds so as to obtain a histogram estimator that flexibly adapts its bin width to the change of density over a range of numeric values. The log-likelihood was used as splitting criterion. Two algorithms were developed: one for one-dimensional data and one for multidimensional data. These newly developed density estimators were applied to several machine learning problems.

Chapters 1 and 2 give an introduction to density estimation and tree building algorithms. Density estimation is explained with a focus on histogram density estimation. A histogram estimator is a density estimation technique that has been used for centuries in statistics. Tree building algorithms discussed in the introductory chapters are decision tree learners as they are used for classification in machine learning.

The work on new algorithms introduced in this thesis is presented in four chapters. Chapters 3 and 4 introduce the univariate tree-based density estimation algorithm TUBE, and discuss its applications. Chapters 5 and 6 cover the multivariate case Multi-TUBE and applications of this algorithm.

Discretization methods in general correspond directly to binning density histogram techniques like equal-width and equal-frequency estimation. TUBE, the new algorithm introduced in Chapter 3, is a binning density estimator and a discretization technique. Histogram estimators are nonparametric. Nonparametric methods are more flexible density estimators than parametric ones and TUBE adapts particularly well to complex distribution functions because it builds a histogram estimator with variable bin length without user input of a smoothing parameter. It can model multiple modalities, abrupt changes in the density, and finds empty areas precisely.

Cross-validation is a standard method to choose between models. In TUBE, cross-validation can be applied incrementally and therefore results in a feasible method to decide on the number of bins. This search for the best binning can

also be viewed as pruning of the tree. Like every histogram, TUBE's binning yields an easily interpretable model of the density function.

In Chapter 4, TUBE discretization was compared with standard unsupervised discretization methods and it was shown that TUBE fits the density function as well or sometimes better than these techniques. It was shown that problems with overfitting due to a small number of instances that lie close together can be solved using heuristics.

Greedy tree-based learning algorithms are known to be very scaleable methods. The greedy solution search in TUBE makes it a density estimation algorithm that is well-behaved in terms of execution time.

In a second application, discussed in Chapter 4, TUBE histograms were used to create probability models in naive Bayes classifiers. Naive Bayes predicts a class value for an instance by computing its class probabilities and by deciding on the class with the highest probability. Part of the overall probability model is the conditional probability (or density) of an attribute value given the class. This is required for each attribute and can be modelled with TUBE histograms. This new method for creating conditional models in naive Bayes was compared with Gaussians and equal-width histograms. It was shown that TUBE density estimation can outperform Gaussians on some datasets but cannot outperform the much simpler equal-width estimator.

Chapters 5 and 6 introduce the multivariate algorithm Multi-TUBE and two applications. Multi-TUBE extends TUBE's binning algorithm to the multivariate case. It splits, like TUBE, the range in a recursive fashion and also places cut points the same way as in TUBE.

Multi-TUBE, described in Chapter 5, considers each attribute for each split but only one attribute at a time. Thus, it performs axis-parallel cuts and the resulting bins have the form of multidimensional rectangles. The splitting criterion is again the log-likelihood and, like TUBE, Multi-TUBE also attempts to identify bins that exhibit a uniform distribution.

When Multi-TUBE creates bins, it ignores irrelevant attributes and it can therefore serve as a feature selection algorithm. Only few unsupervised feature selection algorithms have been developed so far. Note also that a simple method like equal-width estimation, which would consider every attribute for splitting, cannot be applied to multivariate data in practice, because of the high number of bins it would produce.

The multidimensional histogram produced by Multi-TUBE, with its hard split points and non-overlapping bins, is an easy to understand model when

presented as a tree. However, it does not yield a good overview of the distribution of the space and the data into bins.

To achieve this objective, a suitable representation technique was developed. It lists the bins in the order in which they occur in the tree and outputs columns of the most significant measurements—percentage of instances, density and percentage of total volume—using strings of 'X' characters. The columns can be read like a stacked histogram, where the volume—which is the 'width' of the multidimensional histogram is not shown directly as the width of the bin but using a string of 'X's in the corresponding column.

In Chapter 6, a TUBE clusterer for multidimensional data was implemented based on the binning produced by Multi-TUBE. The clustering method is a mode-seeking algorithm and can be classified as a probabilistic clusterer. It was compared with the probabilistic clusterer EM based on the way it finds clusters and its time performance. It was shown that it can find accurate clusters with much less computational effort.

The nature of Multi-TUBE's algorithm means that the TUBE clusterer can cope well with noise. Compared to many other clustering algorithms, one of the advantages of the TUBE clusterer is that it yields an easily comprehensible representation of the clusters it finds. Its output can be represented as a conjunctive statement combined with disjunctions. Although it was found that cross-validation produced too many bins for good results in the clustering application, the algorithm was applied successfully when the number of bins was set by user parameter.

A second application of Multi-TUBE, also discussed in Chapter 6, is positioned in the specialized area of multiple-instance learning, where each example consists of a collection of attribute vectors called a bag of instances. The individual instances in the bag have unknown labels and only the bag has a label assigned to it. Diverse Density is a well known multiple-instance algorithm, which gives one of the best accuracy results on standard multiple-instance test data, but has impractical runtime behaviour. Chapter 6 shows that, with the help of the TUBE clusterer and a new algorithm for mixing two binnings, areas of differing density can be found and used to yield better starting points for the Diverse Density classifier to speed up the learning process.

7.2 Main Contributions

In this thesis, two novel tree-based density estimation methods were introduced and evaluated: TUBE and Multi-TUBE. TUBE is a univariate binning algorithm and Multi-TUBE its adaptation to multivariate data. Experiments showed that the algorithms are able to identify significant components of univariate and multivariate density functions. They define bins as univariate subranges or as multidimensional rectangles respectively, by cutting the space recursively in a greedy fashion into subranges of varying length, with the aim of finding a good piece-wise constant approximation to the density function. Experiments showed that the binning algorithms developed, adapt well to changes in the density of the data, regardless of the complexity of the distribution.

Another important contribution of this thesis is the new clustering algorithm, which it introduces. The TUBE clusterer uses Multi-TUBE's binning to find clusters by first searching for mode bins and connecting them with all their surrounding bins of lower but still significant density. In this way the TUBE clusterer effectively and efficiently finds clusters in the data by identifying 'density bumps' in the multidimensional density estimation function. The shape of the clusters found is flexible enough to accommodate a variety of possible structures in multidimensional data. The clusterer can deal with outliers or noise in data and was shown to have good scalability. Because it has the ability to ignore irrelevant attributes it can cope with datasets of high dimensionality.

It was also shown how univariate density estimation tree learning can automatically determine an appropriate number of bins using the cross-validated log-likelihood. Moreover, it can do so efficiently because trees can be grown by incrementally adding nodes. Very few unsupervised algorithms have this advantage.

It was shown how TUBE and Multi-TUBE can be applied to classification tasks in machine learning. The application of TUBE to naive Bayes showed reasonable results but did not yield an improvement on equal-width discretization. However, the application of Multi-TUBE to multiple-instance learning successfully improved the runtime of the Diverse Density learning algorithm in a significant manner.

It is particularly important to note that, like most greedy algorithms, and specifically other tree learners, TUBE and Multi-TUBE are well-behaved considering computational complexity and thus suitable for use in practical appli-

cations with large datasets and high-dimensional data.

7.3 Repeatability

To ensure that all experiments performed in this thesis are repeatable, the software has been made publicly available from

`http://www.cs.waikato.ac.nz/ml/weka/TUBE`

together with the datasets generated for the clustering application. For each experiment, documentation like the command line program call with the respective parameter settings is provided. Also documented are the program calls for the generation of diagrams and representation techniques (bin lists and bin position overviews).

7.4 Conclusions

There is a close connection between binning density estimators and discretization. Discretization is the term used in machine learning for splitting the range of a numeric attribute into subranges for further transformation of the data. Discretization is mostly employed in classification and many supervised methods have been developed, but only few unsupervised ones. However, as is standard practise in statistics, the binning can also be used to produce a density estimator.

Nonparametric density estimation is more flexible than a parametric approach. Standard unsupervised methods like equal-width and equal-frequency discretization are bound by fixed parameters, namely the bin width and the number of instances per bin respectively. Fayyad & Irani's supervised method [18] employs the entropy values computed from the class labels to flexibly adapt interval boundaries. TUBE and Multi-TUBE are unsupervised discretization algorithms that use the log-likelihood as splitting criterion. This results in density estimators that can closely follow the form of complex density functions. Both algorithms choose variable bin widths and appropriate numbers of instances per bin implicitly. Using a greedy tree-based algorithm to cut the range makes it possible to achieve a runtime behaviour that facilitates the application to practical problems.

Data mining often has to deal with datasets with many instances and many attributes. This provides significant challenges for algorithms. Often attributes are irrelevant and some algorithms are sensitive to this. So far there

are only few unsupervised feature selection algorithms that have been developed. The Multi-TUBE binning procedure, an unsupervised method, selects the next best split by choosing between attributes and thus automatically ignores irrelevant ones.

Many known clustering algorithms include a strong bias regarding the shape of the clusters that can be found. TUBE is very flexible in this respect. Moreover, in the multidimensional case, clusters can be found in subspaces of the features with a different subspace for each cluster. Thus the TUBE clusterer is a subspace clusterer that can find clusters with complex shapes.

Obtaining insight into the structure of multidimensional data can help to analyze its structure and choose the right methods for further analysis. In the application to multiple-instance learning in this thesis, Multi-TUBE was used to find modes in multivariate data and provide the subsequently applied multiple-instance learning algorithm with better starting points to accelerate its search for the underlying concept.

7.5 Future Work

The univariate and the multivariate algorithm developed in this thesis have been applied in very different areas of machine learning. This demonstrates their versatility. Thus it can be expected that there are other application areas and further possibilities for future work.

Naive Bayes Classification Further possibilities should be explored to make TUBE a better estimator when used in Naive Bayes classification. The implementation of a soft boundary could help prevent overfitting.

Enhancing Multivariate Gaussian or Kernel Estimators TUBE histograms could be used to find the position of modes for multi-mode Gaussian estimators or kernel estimators. Kernel estimators have been successfully used as density estimators in naive Bayes classifiers but can exhibit poor runtime behaviour. TUBE histograms could help to build a faster kernel estimator.

Ensemble Learning Techniques Applied to Density Estimation Ensemble learning techniques such as bagging, which have been used to enhance the predictive performance in classification and regression problems by building

an ensemble of trees, could easily be adapted and applied to density estimation based on the tree learners introduced in this thesis.

Feature Selection Applications of Multi-TUBE's inherent feature selection ability could be further explored and its performance compared with that of other unsupervised feature selection methods that have been developed.

Further Clustering Methods It is possible to envisage other clustering algorithms that build on the multidimensional binning found by Multi-TUBE. The clustering analysis of the real-world datasets in Chapter 6 showed that many of them contain one cluster only. The TUBE clusterer is a partitional clusterer. A hierarchical clusterer could be developed that defines a more fine grained clustering for the mode bins found by the existing TUBE clusterer, which would be the next lower level hierarchy. A further possibility would be to use the split tree generated by TUBE to define a hierarchy on the existing clusters.

Finding Outliers in Data There are many applications of clustering and very few could be explored in this work. One further task the TUBE clusterer could be applied to is the identification of outliers in data.

Enhancing the Multivariate Histogram In this thesis a simple representation method was developed to represent the multidimensional histogram constructed by Multi-TUBE. Some possible future work for the expansion of this to an interactive visualization has already been discussed in Chapter 5.

Selecting a Classification Method Mitchell asks in [45]: "What is the relationship between different learning algorithms and which should be used when?". In many cases the answer to this questions could be given if more knowledge about the structure of the data were available. However, gaining information about the data is difficult when it is multidimensional. The analysis of the multidimensional histograms generated by Multi-TUBE could help to produce input for this selection task.

Appendix A

Bin Lists

This appendix contains the bin lists of the three datasets eastwest, musk1 and musk2. The bin lists were generated using the mixing of binnings technique, started with ten bins on each of the two subdatasets. The split into two datasets was performed according to the class value of the instances. The characters `a` and `A` refer to the positive instances and the characters `b` and `B` refer to the negative instances. The bins were then clustered using the TUBE clusterer. Each cluster is listed as one separate bin list. Because a mixing of binnings was performed the bins are ordered according to their difference density (D_{diff}).

A.1 The *eastwest* Dataset

The mixing of binnings applied to the eastwest dataset resulted in 509 bins and nine clusters.

```
Cluster Bin List: 0
Mode Bin: 0

0  :AbbbbbbbbbbB Dns:[XXXXXXXXXX] Ins:[X.........] Vol:[X.........] 3.76%
1  :Abbbb......B Dns:[XXXX......] Ins:[X.........] Vol:[X.........] 2.82%
2  :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
...
...
75 :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
76 :Ba.........A Dns:[X.........] Ins:[X.........] Vol:[X.........] 0.94%
77 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
78 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
79 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
```

```
80 :Baaaaa.....A Dns:[XXX.......] Ins:[X.........] Vol:[X.........] 0.94%
81 :Baaaaaaa...A Dns:[XXXX......] Ins:[X.........] Vol:[X.........] 2.82%
82 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
83 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
84 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
85 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%

Cluster Bin List: 1
Mode bin: 0

0  :AbbbbbbbbbbB Dns:[XXXXXXXXXX] Ins:[X.........] Vol:[X.........] 3.76%
1  :Abbbb......B Dns:[XXXX......] Ins:[X.........] Vol:[X.........] 2.82%
2  :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
...
...
75 :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
76 :Ba.........A Dns:[X.........] Ins:[X.........] Vol:[X.........] 0.94%
77 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
78 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
79 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
80 :Baaaaa.....A Dns:[XXX.......] Ins:[X.........] Vol:[X.........] 0.94%
81 :Baaaaaaa...A Dns:[XXXX......] Ins:[X.........] Vol:[X.........] 2.82%
82 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
83 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
84 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
85 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%

Cluster Bin List: 2
Mode bin: 0

0  :0abbb......B Dns:[XXXX......] Ins:[XX........] Vol:[X.........] inst 10.33%
1  :Abb........B Dns:[XX........] Ins:[X.........] Vol:[X.........] 8.45%
2  :0aaa.......B Dns:[XXXX......] Ins:[X.........] Vol:[X.........] 2.82%
3  :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
...
...
61 :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
62 :Ba.........A Dns:[X.........] Ins:[X.........] Vol:[X.........] 0.94%
63 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
64 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
65 :0bbbbbbaaa.A Dns:[XXXXXXXXXX] Ins:[X.........] Vol:[X.........] 3.76%
66 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
67 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
```

```
Cluster Bin List: 3
Mode bin: 0

0  :Abbb.......B Dns:[XXX.......] Ins:[X.........] Vol:[X.........] 7.04%
1  :0aab.......B Dns:[XXXX......] Ins:[X.........] Vol:[X.........] 2.35%
2  :Abb........B Dns:[XX........] Ins:[X.........] Vol:[X.........] 7.04%
3  :0aaaaab....B Dns:[XXXXXXXX..] Ins:[X.........] Vol:[X.........] 2.82%
4  :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
...
...
64 :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
65 :Ba.........A Dns:[X.........] Ins:[X.........] Vol:[X.........] 0.94%
66 :0ba........A Dns:[X.........] Ins:[X.........] Vol:[X.........] 2.82%
67 :0ba........A Dns:[X.........] Ins:[X.........] Vol:[X.........] 2.35%
68 :0baa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 2.82%
69 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
70 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
71 :0bbaaaa....A Dns:[XXXXX.....] Ins:[X.........] Vol:[X.........] 6.57%
72 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
73 :Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%

Cluster Bin List: 4
Mode bin: 0

0  :Abb........B Dns:[XX........] Ins:[X.........] Vol:[X.........] 1.41%
1  :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
...
...
19 :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%

Cluster Bin List: 5
Mode bin: 0

0  :Abb........B Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
1  :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
...
...
23 :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
24 :Baa........A Dns:[X.........] Ins:[X.........] Vol:[X.........] 0.94%
25 :Baaaaa.....A Dns:[XXX.......] Ins:[X.........] Vol:[X.........] 0.94%

Cluster Bin List: 6
Mode bin: 0
```

```
0  :Abb........B Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
1  :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
...
...
24 :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
25 :Baa........A Dns:[X.........] Ins:[X.........] Vol:[X.........] 0.94%

Cluster Bin List: 7

0  :0aaa.......B Dns:[XXXX......] Ins:[X.........] Vol:[X.........] 2.82%
1  :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
...
...
20 :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%

Cluster Bin List: 8
Mode bin: 0

0  :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
...
...
91 :-..........- Dns:[          ] Ins:[          ] Vol:[X.........] 0.0%
92 :Ba.........A Dns:[X.........] Ins:[X.........] Vol:[X.........] 0.94%
93 :0ba........A Dns:[X.........] Ins:[X.........] Vol:[X.........] 2.82%
94 :0ba........A Dns:[X.........] Ins:[X.........] Vol:[X.........] 2.35%
95 :Baa........A Dns:[X.........] Ins:[X.........] Vol:[X.........] 0.94%
96 :0baa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 2.82%
97 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
98 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
99 :Baaa.......A Dns:[XX........] Ins:[X.........] Vol:[X.........] 0.94%
100:0bbbbbbaaa.A Dns:[XXXXXXXXXX] Ins:[X.........] Vol:[X.........] 3.76%
101:0bbaaaa....A Dns:[XXXXX.....] Ins:[X.........] Vol:[X.........] 6.57%
102:Baaaaa.....A Dns:[XXX.......] Ins:[X.........] Vol:[X.........] 0.94%
103:Baaaaa.....A Dns:[XXX.......] Ins:[X.........] Vol:[X.........] 0.94%
104:Baaaaaaa...A Dns:[XXXX......] Ins:[X.........] Vol:[X.........] 2.82%
105:Baaaaaaa...A Dns:[XXXX......] Ins:[X.........] Vol:[X.........] 2.82%
106:Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
107:Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
108:Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
109:Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
110:Baaaaaaaaa.A Dns:[XXXXXX....] Ins:[X.........] Vol:[X.........] 1.88%
```

A.2 The *musk1* Dataset

The clusterer found only one cluster in the binning of the musk1 dataset. The bin list of this cluster contains 584 bins.

```
Cluster Bin List: 0
Mode bin: 0

0  :0aaaabbbbbbB Dns:[XXXXXXXXXX] Ins:[XX........] Vol:[<1E-17....] 15.13%
1  :0a.........B Dns:[X.........] Ins:[X.........] Vol:[<1E-15....] 3.15%
2  :Ab.........B Dns:[<0.1......] Ins:[X.........] Vol:[<1E-15....] 0.21%
3  :0a.........B Dns:[X.........] Ins:[X.........] Vol:[<1E-15....] 1.26%
4  :0a.........B Dns:[<0.1......] Ins:[X.........] Vol:[<1E-14....] 8.4%
5  :Ab.........B Dns:[<0.01.....] Ins:[X.........] Vol:[<1E-14....] 0.21%
6  :Ab.........B Dns:[<0.001....] Ins:[X.........] Vol:[<1E-14....] 0.21%
7  :0a.........B Dns:[<0.01.....] Ins:[X.........] Vol:[<1E-13....] 1.68%
8  :Ab.........B Dns:[<0.001....] Ins:[X.........] Vol:[<1E-13....] 0.21%
9  :0a.........B Dns:[<0.001....] Ins:[X.........] Vol:[<1E-13....] 0.63%
10 :0a.........B Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-11....] 3.36%
11 :0a.........B Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-12....] 0.42%
12 :Ab.........B Dns:[<0.00001..] Ins:[X.........] Vol:[<1E-11....] 0.21%
13 :Ab.........B Dns:[<0.000001.] Ins:[X.........] Vol:[<1E-11....] 0.21%
14 :0a.........B Dns:[<0.00001..] Ins:[X.........] Vol:[<1E-11....] 0.42%
15 :0a.........B Dns:[<0.000001.] Ins:[X.........] Vol:[<1E-10....] 0.63%
16 :0a.........B Dns:[<0.000001.] Ins:[X.........] Vol:[<1E-10....] 0.42%
17 :Ab.........B Dns:[<1E-9.....] Ins:[X.........] Vol:[<1E-9.....] 0.21%
18 :0a.........B Dns:[<1E-8.....] Ins:[X.........] Vol:[<1E-9.....] 1.26%
19 :0a.........B Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.000001.] 1.89%
20 :Ab.........B Dns:[<1E-10....] Ins:[X.........] Vol:[<0.000001.] 0.63%
21 :Ab.........B Dns:[<1E-10....] Ins:[X.........] Vol:[<1E-8.....] 0.21%
22 :0a.........B Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.000001.] 2.94%
23 :0a.........B Dns:[<1E-12....] Ins:[X.........] Vol:[<0.0001...] 0.42%
24 :-.........- Dns:[          ] Ins:[          ] Vol:[<1E-17....] 0.0%
...
...
522:-.........- Dns:[          ] Ins:[          ] Vol:[XXXXXXXXX.] 0.0%
523:Ba.........A Dns:[<1E-16....] Ins:[X.........] Vol:[<0.1......] 0.21%
524:Ba.........A Dns:[<1E-15....] Ins:[X.........] Vol:[<0.1......] 0.21%
525:Ba.........A Dns:[<1E-15....] Ins:[X.........] Vol:[<0.1......] 0.21%
526:0b.........A Dns:[<1E-13....] Ins:[X.........] Vol:[<0.001....] 0.84%
527:Ba.........A Dns:[<1E-13....] Ins:[X.........] Vol:[<0.001....] 0.42%
528:Ba.........A Dns:[<1E-13....] Ins:[X.........] Vol:[<0.001....] 0.21%
529:Ba.........A Dns:[<1E-13....] Ins:[X.........] Vol:[<0.001....] 0.21%
530:Ba.........A Dns:[<1E-13....] Ins:[X.........] Vol:[<0.001....] 0.21%
531:Ba.........A Dns:[<1E-13....] Ins:[X.........] Vol:[<0.001....] 0.21%
```

```
532:Ba.........A Dns:[<1E-13....] Ins:[X.........] Vol:[<0.001....] 0.21%
533:Ba.........A Dns:[<1E-13....] Ins:[X.........] Vol:[<0.001....] 0.21%
534:Ba.........A Dns:[<1E-13....] Ins:[X.........] Vol:[<0.001....] 0.84%
535:0b.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.0001...] 1.05%
536:Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.001....] 0.84%
537:Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.0001...] 0.21%
538:Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.0001...] 0.21%
539:Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.0001...] 0.21%
540:Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.0001...] 0.42%
541:Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.00001..] 0.21%
542:0b.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.00001..] 0.63%
543:Ba.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.00001..] 0.21%
544:Ba.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.00001..] 0.42%
545:Ba.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.0001...] 1.47%
546:Ba.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.00001..] 0.21%
547:0b.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.00001..] 2.1%
548:Ba.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.00001..] 1.05%
549:0b.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.000001.] 5.88%
550:Ba.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.000001.] 0.21%
551:0b.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.000001.] 0.63%
552:Ba.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.000001.] 0.21%
553:Ba.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<1E-8.....] 0.21%
554:Ba.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.000001.] 1.68%
555:0b.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<1E-9.....] 1.05%
556:Ba.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<1E-9.....] 0.21%
557:Ba.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<1E-9.....] 0.21%
558:Ba.........A Dns:[<1E-8.....] Ins:[X.........] Vol:[<1E-9.....] 0.21%
559:Ba.........A Dns:[<1E-8.....] Ins:[X.........] Vol:[<1E-10....] 0.21%
560:Ba.........A Dns:[<1E-8.....] Ins:[X.........] Vol:[<1E-10....] 0.21%
561:Ba.........A Dns:[<1E-8.....] Ins:[X.........] Vol:[<1E-9.....] 0.42%
562:0b.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<1E-9.....] 1.47%
563:Ba.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<1E-10....] 0.63%
564:0b.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<1E-10....] 1.05%
565:Ba.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<1E-10....] 0.21%
566:Ba.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<1E-10....] 0.63%
567:0b.........A Dns:[<0.00001..] Ins:[XX........] Vol:[<1E-10....] 13.03%
568:0b.........A Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-12....] 1.47%
569:Ba.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<1E-11....] 0.21%
570:0b.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<1E-11....] 0.63%
571:Ba.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<1E-11....] 0.21%
572:Ba.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<1E-11....] 0.21%
573:Ba.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<1E-11....] 0.63%
574:Ba.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<1E-11....] 0.42%
575:Ba.........A Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-12....] 0.21%
```

```
576:Ba.........A Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-12....] 0.21%
577:Ba.........A Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-13....] 0.21%
578:Ba.........A Dns:[<0.001....] Ins:[X.........] Vol:[<1E-13....] 0.42%
579:Ba.........A Dns:[<0.001....] Ins:[X.........] Vol:[<1E-13....] 0.63%
580:Ba.........A Dns:[<0.01.....] Ins:[X.........] Vol:[<1E-14....] 0.42%
581:0b.........A Dns:[X.........] Ins:[X.........] Vol:[<1E-15....] 7.35%
582:Ba.........A Dns:[<0.1......] Ins:[X.........] Vol:[<1E-16....] 0.21%
583:Ba.........A Dns:[X.........] Ins:[X.........] Vol:[<1E-16....] 0.63%
```

A.3 The *musk2* Dataset

The mixing of binnings resulted in 722 bins for the musk2 dataset. The TUBE clusterer found two clusters.

```
Cluster Bin List: 0
Mode Bin: 0

0  :0abbbbbbbbbB Dns:[XXXX......] Ins:[X.........] Vol:[<1E-15....] 1.85%
1  :0aaaaabbbbbB Dns:[XXXXXXXXXX] Ins:[XX........] Vol:[<1E-14....] 19.38%
2  :Abbb.......B Dns:[X.........] Ins:[<0.1......] Vol:[<1E-16....] 0.06%
3  :0ab........B Dns:[XX........] Ins:[X.........] Vol:[<1E-15....] 0.41%
4  :0ab........B Dns:[XXX.......] Ins:[X.........] Vol:[<1E-16....] 0.2%
5  :Ab.........B Dns:[X.........] Ins:[<0.1......] Vol:[<1E-15....] 0.02%
6  :0a.........B Dns:[X.........] Ins:[X.........] Vol:[<1E-14....] 0.29%
7  :Ab.........B Dns:[X.........] Ins:[X.........] Vol:[<1E-14....] 0.11%
8  :0a.........B Dns:[X.........] Ins:[X.........] Vol:[<1E-13....] 0.61%
9  :0a.........B Dns:[X.........] Ins:[<0.1......] Vol:[<1E-14....] 0.09%
10 :0a.........B Dns:[<0.1......] Ins:[<0.1......] Vol:[<1E-14....] 0.05%
11 :0a.........B Dns:[<0.1......] Ins:[X.........] Vol:[<1E-12....] 0.8 %
12 :0a.........B Dns:[<0.01.....] Ins:[X.........] Vol:[<1E-12....] 0.3%
13 :0a.........B Dns:[<0.1......] Ins:[X.........] Vol:[<1E-12....] 0.15%
14 :0a.........B Dns:[<0.01.....] Ins:[<0.1......] Vol:[<1E-13....] 0.08%
15 :0a.........B Dns:[<0.1......] Ins:[XX........] Vol:[<1E-11....] 10.64%
16 :Ab.........B Dns:[<0.001....] Ins:[X.........] Vol:[<1E-11....] 0.17%
17 :0a.........B Dns:[<0.001....] Ins:[X.........] Vol:[<1E-11....] 0.35%
18 :0a.........B Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-10....] 0.33%
19 :0a.........B Dns:[<0.001....] Ins:[X.........] Vol:[<1E-10....] 1.68%
20 :0a.........B Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-9.....] 0.06%
21 :0a.........B Dns:[<0.001....] Ins:[X.........] Vol:[<1E-10....] 0.95%
22 :Ab.........B Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-9.....] 0.09%
23 :0a.........B Dns:[<0.000001.] Ins:[<0.1......] Vol:[<1E-9.....] 0.05%
24 :0a.........B Dns:[<0.000001.] Ins:[<0.1......] Vol:[<0.000001.] 0.05%
25 :Ab.........B Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.09%
26 :0a.........B Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.09%
```

```
27 :-.........- Dns:[          ] Ins:[          ] Vol:[<1E-16....] 0.0%
...
...
269:-.........- Dns:[          ] Ins:[          ] Vol:[XXXXXXXXX.] 0.0%
270:Ba.........A Dns:[<1E-15....] Ins:[<0.1......] Vol:[X.........] 0.08%
271:Ba.........A Dns:[<1E-14....] Ins:[X.........] Vol:[X.........] 0.14%
272:Ba.........A Dns:[<1E-14....] Ins:[<0.1......] Vol:[<0.1......] 0.03%
273:Ba.........A Dns:[<1E-14....] Ins:[<0.1......] Vol:[<0.1......] 0.06%
274:Ba.........A Dns:[<1E-14....] Ins:[<0.1......] Vol:[<0.1......] 0.06%
275:Ba.........A Dns:[<1E-13....] Ins:[<0.1......] Vol:[<0.1......] 0.02%
276:Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.1......] 0.11%
277:Ba.........A Dns:[<1E-12....] Ins:[<0.1......] Vol:[<0.01.....] 0.06%
278:Ba.........A Dns:[<1E-12....] Ins:[<0.1......] Vol:[<0.01.....] 0.03%
279:Ba.........A Dns:[<1E-12....] Ins:[<0.1......] Vol:[<0.001....] 0.02%
280:Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.1......] 0.18%
281:Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.1......] 0.18%
282:Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.01.....] 0.17%
283:Ba.........A Dns:[<1E-12....] Ins:[<0.1......] Vol:[<0.001....] 0.03%
284:Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.01.....] 0.35%
285:Ba.........A Dns:[<1E-12....] Ins:[<0.1......] Vol:[<0.001....] 0.03%
286:Ba.........A Dns:[<1E-12....] Ins:[<0.1......] Vol:[<0.001....] 0.06%
287:Ba.........A Dns:[<1E-11....] Ins:[<0.1......] Vol:[<0.001....] 0.02%
288:Ba.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.001....] 0.15%
289:Ba.........A Dns:[<1E-11....] Ins:[<0.1......] Vol:[<0.001....] 0.03%
290:Ba.........A Dns:[<1E-11....] Ins:[<0.1......] Vol:[<0.001....] 0.08%
291:Ba.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.001....] 0.52%
292:Ba.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.001....] 0.11%
293:Ba.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.001....] 0.17%
294:Ba.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.001....] 0.83%
295:Ba.........A Dns:[<1E-10....] Ins:[<0.1......] Vol:[<0.0001...] 0.02%
296:0b.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.001....] 1.56%
297:Ba.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.001....] 0.21%
298:Ba.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.0001...] 0.17%
299:Ba.........A Dns:[<1E-10....] Ins:[<0.1......] Vol:[<0.00001..] 0.03%
300:Ba.........A Dns:[<1E-10....] Ins:[<0.1......] Vol:[<0.00001..] 0.03%
301:Ba.........A Dns:[<1E-10....] Ins:[<0.1......] Vol:[<0.0001...] 0.05%
302:Ba.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.0001...] 0.26%
303:0b.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.001....] 1.21%
304:Ba.........A Dns:[<1E-10....] Ins:[<0.1......] Vol:[<0.00001..] 0.02%
305:Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.09%
306:Ba.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.0001...] 0.15%
307:Ba.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.0001...] 0.21%
308:0b.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.00001..] 0.27%
309:Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.05%
```

```
310:0b.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.0001...] 2.3%
311:Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.05%
312:Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.05%
313:0b.........A Dns:[<1E-8.....] Ins:[X.........] Vol:[<0.0001...] 1.94%
314:Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.06%
315:Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.08%
316:Ba.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.00001..] 0.11%
317:Ba.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.0001...] 0.61%
318:Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.000001.] 0.03%
319:Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.000001.] 0.08%
320:Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.000001.] 0.03%
321:Ba.........A Dns:[<1E-8.....] Ins:[<0.1......] Vol:[<0.000001.] 0.08%
322:Ba.........A Dns:[<1E-8.....] Ins:[X.........] Vol:[<0.00001..] 0.8%
323:Ba.........A Dns:[<1E-8.....] Ins:[X.........] Vol:[<0.00001..] 0.15%
324:Ba.........A Dns:[<1E-8.....] Ins:[<0.1......] Vol:[<0.000001.] 0.02%
325:0b.........A Dns:[<1E-8.....] Ins:[X.........] Vol:[<0.00001..] 2.29%
326:0b.........A Dns:[<1E-8.....] Ins:[X.........] Vol:[<0.00001..] 0.39%
327:Ba.........A Dns:[<0.000001.] Ins:[<0.1......] Vol:[<0.000001.] 0.05%
328:Ba.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<0.000001.] 0.27%
329:Ba.........A Dns:[<0.000001.] Ins:[<0.1......] Vol:[<0.000001.] 0.06%
330:Ba.........A Dns:[<0.000001.] Ins:[<0.1......] Vol:[<0.000001.] 0.08%
331:Ba.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<0.000001.] 0.15%
332:0b.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<0.000001.] 1.29%
333:0b.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<0.000001.] 1.39%
334:0b.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<0.000001.] 1.94%
335:Ba.........A Dns:[<0.000001.] Ins:[<0.1......] Vol:[<1E-9.....] 0.05%
336:0b.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<0.000001.] 0.71%
337:Ba.........A Dns:[<0.000001.] Ins:[<0.1......] Vol:[<1E-9.....] 0.02%
338:Ba.........A Dns:[<0.000001.] Ins:[<0.1......] Vol:[<1E-9.....] 0.05%
339:Ba.........A Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-9.....] 0.06%
340:Ba.........A Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-9.....] 0.02%
341:Ba.........A Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-9.....] 0.08%
342:Ba.........A Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-9.....] 0.09%
343:0b.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<0.000001.] 0.7%
344:0b.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<0.000001.] 2.52%
345:Ba.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<1E-9.....] 0.15%
346:0b.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<1E-8.....] 0.85%
347:0b.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<0.000001.] 1.38%
348:Ba.........A Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-10....] 0.02%
349:0b.........A Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-10....] 0.32%
350:Ba.........A Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-10....] 0.02%
351:Ba.........A Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-9.....] 0.06%
352:0b.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<1E-8.....] 2.68%
353:Ba.........A Dns:[<0.0001...] Ins:[<0.1......] Vol:[<1E-10....] 0.05%
```

```
354:0b.........A Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-9.....] 0.44%
355:0b.........A Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-9.....] 1.09%
356:0b.........A Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-10....] 0.24%
357:0b.........A Dns:[<0.001....] Ins:[X.........] Vol:[<1E-10....] 0.39%
358:Ba.........A Dns:[<0.0001...] Ins:[<0.1......] Vol:[<1E-11....] 0.05%
359:0b.........A Dns:[<0.001....] Ins:[X.........] Vol:[<1E-9.....] 2.93%
360:0b.........A Dns:[<0.001....] Ins:[X.........] Vol:[<1E-10....] 1.18%
361:0b.........A Dns:[<0.001....] Ins:[X.........] Vol:[<1E-9.....] 7.29%
362:Ba.........A Dns:[<0.001....] Ins:[<0.1......] Vol:[<1E-12....] 0.02%
363:0b.........A Dns:[<0.001....] Ins:[X.........] Vol:[<1E-10....] 2.58%
364:Ba.........A Dns:[<0.001....] Ins:[<0.1......] Vol:[<1E-12....] 0.02%
365:Ba.........A Dns:[<0.001....] Ins:[X.........] Vol:[<1E-11....] 0.18%
366:0b.........A Dns:[<0.01.....] Ins:[X.........] Vol:[<1E-11....] 0.82%
367:Ba.........A Dns:[<0.01.....] Ins:[X.........] Vol:[<1E-11....] 0.77%
368:Ba.........A Dns:[<0.01.....] Ins:[<0.1......] Vol:[<1E-12....] 0.03%
369:Ba.........A Dns:[<0.01.....] Ins:[<0.1......] Vol:[<1E-13....] 0.02%
370:0b.........A Dns:[<0.1......] Ins:[X.........] Vol:[<1E-12....] 1.18%
371:Ba.........A Dns:[<0.1......] Ins:[<0.1......] Vol:[<1E-14....] 0.03%
372:Ba.........A Dns:[X.........] Ins:[<0.1......] Vol:[<1E-14....] 0.02%
373:0b.........A Dns:[X.........] Ins:[X.........] Vol:[<1E-15....] 0.11%
374:0b.........A Dns:[X.........] Ins:[X.........] Vol:[<1E-13....] 4.88%
375:0b.........A Dns:[X.........] Ins:[X.........] Vol:[<1E-14....] 0.17%

Cluster Bin List: 1
Mode Bin: 0

0   : 0a.........B Dns:[<0.01.....] Ins:[X.........] Vol:[<1E-12....] 0.32%
1   : Ab.........B Dns:[<0.001....] Ins:[X.........] Vol:[<1E-11....] 0.17%
2   : 0a.........B Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-10....] 0.18%
3   : 0a.........B Dns:[<0.001....] Ins:[X.........] Vol:[<1E-10....] 1.68%
4   : 0a.........B Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-9.....] 0.06%
5   : Ab.........B Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-9.....] 0.09%
6   : 0a.........B Dns:[<0.000001.] Ins:[<0.1......] Vol:[<0.000001.] 0.05%
7   : Ab.........B Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.09%
8   : 0a.........B Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.09%
9   : -..........- Dns:[          ] Ins:[          ] Vol:[<1E-16....] 0.0%
...
...
245: -..........- Dns:[          ] Ins:[          ] Vol:[XXXXXXXXX.] 0.0%
246: Ba.........A Dns:[<1E-15....] Ins:[<0.1......] Vol:[X.........] 0.08%
247: Ba.........A Dns:[<1E-14....] Ins:[X.........] Vol:[X.........] 0.14%
248: Ba.........A Dns:[<1E-14....] Ins:[<0.1......] Vol:[<0.1......] 0.03%
249: Ba.........A Dns:[<1E-14....] Ins:[<0.1......] Vol:[<0.1......] 0.06%
250: Ba.........A Dns:[<1E-14....] Ins:[<0.1......] Vol:[<0.1......] 0.06%
```

```
251: Ba.........A Dns:[<1E-13....] Ins:[<0.1......] Vol:[<0.1......] 0.02%
252: Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.1......] 0.11%
253: Ba.........A Dns:[<1E-12....] Ins:[<0.1......] Vol:[<0.01.....] 0.06%
254: Ba.........A Dns:[<1E-12....] Ins:[<0.1......] Vol:[<0.01.....] 0.03%
255: Ba.........A Dns:[<1E-12....] Ins:[<0.1......] Vol:[<0.001....] 0.02%
256: Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.1......] 0.18%
257: Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.1......] 0.18%
258: Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.01.....] 0.17%
259: Ba.........A Dns:[<1E-12....] Ins:[<0.1......] Vol:[<0.001....] 0.03%
260: Ba.........A Dns:[<1E-12....] Ins:[X.........] Vol:[<0.01.....] 0.35%
261: Ba.........A Dns:[<1E-12....] Ins:[<0.1......] Vol:[<0.001....] 0.03%
262: Ba.........A Dns:[<1E-12....] Ins:[<0.1......] Vol:[<0.001....] 0.06%
263: Ba.........A Dns:[<1E-11....] Ins:[<0.1......] Vol:[<0.001....] 0.02%
264: Ba.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.001....] 0.15%
265: Ba.........A Dns:[<1E-11....] Ins:[<0.1......] Vol:[<0.001....] 0.03%
266: Ba.........A Dns:[<1E-11....] Ins:[<0.1......] Vol:[<0.001....] 0.06%
267: Ba.........A Dns:[<1E-11....] Ins:[<0.1......] Vol:[<0.001....] 0.08%
268: Ba.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.001....] 0.52%
269: Ba.........A Dns:[<1E-11....] Ins:[X.........] Vol:[<0.001....] 0.11%
270: Ba.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.001....] 0.17%
271: Ba.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.001....] 0.83%
272: Ba.........A Dns:[<1E-10....] Ins:[<0.1......] Vol:[<0.0001...] 0.02%
273: 0b.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.001....] 1.56%
274: Ba.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.001....] 0.21%
275: Ba.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.0001...] 0.17%
276: Ba.........A Dns:[<1E-10....] Ins:[<0.1......] Vol:[<0.00001..] 0.03%
277: Ba.........A Dns:[<1E-10....] Ins:[<0.1......] Vol:[<0.00001..] 0.03%
278: Ba.........A Dns:[<1E-10....] Ins:[<0.1......] Vol:[<0.0001...] 0.05%
279: Ba.........A Dns:[<1E-10....] Ins:[X.........] Vol:[<0.0001...] 0.26%
280: 0b.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.001....] 1.21%
281: Ba.........A Dns:[<1E-10....] Ins:[<0.1......] Vol:[<0.00001..] 0.02%
282: Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.09%
283: Ba.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.0001...] 0.15%
284: Ba.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.0001...] 0.21%
285: 0b.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.00001..] 0.27%
286: Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.06%
287: Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.05%
288: 0b.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.0001...] 2.3%
289: Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.05%
290: Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.05%
291: 0b.........A Dns:[<1E-8.....] Ins:[X.........] Vol:[<0.0001...] 1.94%
292: Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.06%
293: 0b.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.00001..] 0.44%
294: Ba.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.00001..] 0.33%
```

```
295: Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.00001..] 0.08%
296: Ba.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.00001..] 0.11%
297: Ba.........A Dns:[<1E-9.....] Ins:[X.........] Vol:[<0.0001...] 0.61%
298: 0b.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<1E-8.....] 0.11%
299: Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.000001.] 0.03%
300: Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.000001.] 0.08%
301: Ba.........A Dns:[<1E-9.....] Ins:[<0.1......] Vol:[<0.000001.] 0.03%
302: Ba.........A Dns:[<1E-8.....] Ins:[<0.1......] Vol:[<0.000001.] 0.08%
303: Ba.........A Dns:[<1E-8.....] Ins:[X.........] Vol:[<0.00001..] 0.8%
304: Ba.........A Dns:[<1E-8.....] Ins:[X.........] Vol:[<0.00001..] 0.15%
305: Ba.........A Dns:[<1E-8.....] Ins:[<0.1......] Vol:[<0.000001.] 0.02%
306: Ba.........A Dns:[<1E-8.....] Ins:[<0.1......] Vol:[<0.000001.] 0.02%
307: 0b.........A Dns:[<1E-8.....] Ins:[X.........] Vol:[<0.00001..] 2.29%
308: Ba.........A Dns:[<0.000001.] Ins:[<0.1......] Vol:[<0.000001.] 0.05%
309: Ba.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<0.000001.] 0.27%
310: Ba.........A Dns:[<0.000001.] Ins:[<0.1......] Vol:[<0.000001.] 0.06%
311: Ba.........A Dns:[<0.000001.] Ins:[<0.1......] Vol:[<0.000001.] 0.08%
312: Ba.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<0.000001.] 0.15%
313: Ba.........A Dns:[<0.000001.] Ins:[<0.1......] Vol:[<1E-9.....] 0.03%
314: Ba.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<0.000001.] 0.86%
315: 0b.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<0.000001.] 1.29%
316: 0b.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<0.000001.] 1.39%
317: 0b.........A Dns:[<0.000001.] Ins:[X.........] Vol:[<0.000001.] 1.94%
318: Ba.........A Dns:[<0.000001.] Ins:[<0.1......] Vol:[<1E-9.....] 0.05%
319: 0b.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<0.000001.] 0.71%
320: Ba.........A Dns:[<0.000001.] Ins:[<0.1......] Vol:[<1E-9.....] 0.02%
321: Ba.........A Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-9.....] 0.06%
322: Ba.........A Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-9.....] 0.02%
323: Ba.........A Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-9.....] 0.08%
324: Ba.........A Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-9.....] 0.09%
325: 0b.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<0.000001.] 0.7%
326: Ba.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<1E-9.....] 0.15%
327: 0b.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<0.000001.] 1.38%
328: 0b.........A Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-10....] 0.32%
329: Ba.........A Dns:[<0.00001..] Ins:[<0.1......] Vol:[<1E-10....] 0.03%
330: Ba.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<1E-9.....] 0.15%
331: 0b.........A Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-9.....] 0.45%
332: 0b.........A Dns:[<0.00001..] Ins:[X.........] Vol:[<1E-8.....] 2.68%
333: Ba.........A Dns:[<0.0001...] Ins:[<0.1......] Vol:[<1E-10....] 0.05%
334: 0b.........A Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-9.....] 0.44%
335: 0b.........A Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-9.....] 1.09%
336: 0b.........A Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-10....] 0.24%
337: 0b.........A Dns:[<0.0001...] Ins:[X.........] Vol:[<1E-10....] 0.35%
338: 0b.........A Dns:[<0.001....] Ins:[X.........] Vol:[<1E-9.....] 2.93%
```

```
339: Ob.........A Dns:[<0.001....] Ins:[X.........] Vol:[<1E-10....] 1.18%
340: Ba.........A Dns:[<0.001....] Ins:[<0.1......] Vol:[<1E-12....] 0.02%
341: Ob.........A Dns:[<0.001....] Ins:[X.........] Vol:[<1E-10....] 2.58%
342: Ob.........A Dns:[<0.01.....] Ins:[X.........] Vol:[<1E-11....] 0.82%
343: Ob.........A Dns:[<0.01.....] Ins:[X.........] Vol:[<1E-11....] 1.38%
344: Ba.........A Dns:[<0.01.....] Ins:[X.........] Vol:[<1E-11....] 0.77%
345: Ba.........A Dns:[<0.01.....] Ins:[<0.1......] Vol:[<1E-12....] 0.03%
346: Ba.........A Dns:[<0.1......] Ins:[<0.1......] Vol:[<1E-14....] 0.03%
347: Ob.........A Dns:[X.........] Ins:[X.........] Vol:[<1E-15....] 0.11%
```

Appendix B

Command Line Program Calls

This document is a short user guide for the programs developed in the thesis **Tree-based Density Estimation: Algorithms and Applications** [**?**]. It lists for each application introduced in the thesis relevant examples of java program calls of the corresponding classes from the command line. All programs for this thesis have been implemented in Java and are based on the WEKA machine learning tool set [68]. (Note, all command lines given in this appendix perform only simple 10-fold cross-validation but most experiments performed for the work in this thesis used 10×10 cross-validation using the WEKA experiment environment.)

Appendix C

Program Calls Used in Application Discretization

C.1 Drawing Histograms

Drawing a 10 Bin Equal-Width Histogram Output data for gnuplot to print an equal-width histogram with 10 bins (`-B 10`); the input file is `gauss.arff` (`-i gauss.arff`) and the output file suffix is `gauss-B10` (`-X gauss-B10`).

```
java weka.estimators.EqualWidthEstimator -i gauss.arff -B 10 \
-V 8 -X gauss-B10

[plot with gnuplot]
plot 'gauss-B10-0EW.hist' title "EW 10 bins" with lines
```

Drawing a 10 Bin Equal-Frequency Histogram

```
java weka.estimators.EqualFrequencyEstimator -i gauss.arff -B 10 \
-V 8 -X gauss-B10

[plot with gnuplot]
plot 'gauss-B10-0EF.hist' title "EF 10 bins" with lines
```

Drawing a TUBE Histogram with a fixed number of bins

```
java weka.estimators.TUBEstimator -i gauss.arff -B 10 \
-V 8 -X gauss-B10

[plot with gnuplot]
plot 'gauss-B10-V8-0LL.hist' title "TUBE 10 bins" with lines
```

Drawing a TUBE Histogram, TUBE computes the number of bins using cross-validation

```
java weka.estimators.TUBEstimator -i gauss.arff -B 10 \
-V 8 -X gauss

[plot with gnuplot]
plot 'gauss-0LL.hist' title "TUBE CV" with lines
```

C.2 Discretizing

(TUBE) TUBE discretization with cross-validation for the number of bins (`default`).

```
java weka.estimators.TUBEstimator -i iris01.arff
```

(EW-10) Equal-width discretization with ten bins.

```
java weka.estimators.EqualWidthEstimator -i iris01.arff -B 10
```

(EWcvB) Equal-width discretization with cross-validation for the number of bins (`- Y`); the maximum number of bins is set to 100 (`-B 100`).

```
java weka.estimators.EqualWidthEstimator -i iris01.arff -Y -B 100
```

(EWcvBO) Equal-width discretization with cross-validation for the origin of the bins (`- Z`) and the number of bins (`- Y`); the maximum number of bins is set to 100 (`-B 100`).

```
java weka.estimators.EqualWidthEstimator -i iris01.arff -Y -Z -B 100
```

(EF-10) Equal-frequency discretization with ten bins.

```
java weka.estimators.EqualFrequencyEstimator -i iris01.arff -B 10
```

Appendix D

Program Calls Used in Application Naive Bayes

Introduction: Naive Bayes classifier used with varying estimators In WEKA the standard implementation of a naive Bayes classifier is the class NaiveBayes. NaiveBayes uses Gaussian distributions for density estimation. The class NaiveBayesParametrized substitutes the Gaussian estimators with an estimator given in the parameter `-E`.

```
java weka.classifiers.bayes.NaiveBayesParametrized -t iris.arff \
-E "weka.estimators.EqualWidthEstimator -B 10"
```

(Gauss) Naive Bayes classifier using Gaussian distribution.

```
java weka.classifiers.bayes.NaiveBayes -t iris.arff
```

(TUBE-CV) Naive Bayes classifier using TUBE histograms; the number of bins is cross-validated (`default`); the maximum number of bins is 100 (`-B 100`).

```
java weka.classifiers.bayes.NaiveBayesParametrized -t iris.arff \
-E "weka.estimators.TUBEstimator -B 100"
```

(EW-CV) Naive Bayes classifier using equal-width histograms; the number of bins is cross-validated (`-Z`); the maximum number of bins is 100 (`-B 100`).

```
java weka.classifiers.bayes.NaiveBayesParametrized -t iris.arff \
-E "weka.estimators.EqualWidthEstimator -Z -B 100"
```

(TUBE-02) Naive Bayes classifier using TUBE histograms; TUBE with the cut distance set to 0.1 (`-Z 0.1`) and the minimal bin width set to 0.2 (`-L -U 2`); the number of bins is cross-validated (`default`); the maximum number of bins is 100 (`-B 100`).

```
java weka.classifiers.bayes.NaiveBayesParametrized -t iris.arff \
-E "weka.estimators.TUBEstimator -L -Z 0.1 -U 2 -B 100 "
```

(TUBE-15) Naive Bayes classifier using TUBE histograms; TUBE with the cut distance set to 0.1 (`-Z 0.1`) and the minimal bin width set to 0.2 (`-L -U 2`); the number of bins is cross-validated (`default`); the maximum number of bins is 15 (`-B 15`).

```
java weka.classifiers.bayes.NaiveBayesParametrized -t iris.arff \
-E "weka.estimators.TUBEstimator -L -Z 0.1 -U 2 -B 15"
```

(TUBE-EW) Naive Bayes classifier using TUBE histograms; TUBE estimatior with the cut distance and the minimal bin width set using equal-width heuristic (`-L -U 4`); the number of bins is cross-validated (`default`); the maximum number of bins is 100 (`-B 100`).

```
java weka.classifiers.bayes.NaiveBayesParametrized -t iris.arff \
-E "weka.estimators.TUBEstimator -L -U 4 -B 100 "
```

(EW-15) Naive Bayes classifier using equal-width histograms; the number of bins is fixed to 15 (`-B 15`).

```
java  weka.classifiers.bayes.NaiveBayesParametrized -t iris.arff \
-E "weka.estimators.EqualWidthEstimator -B 15"
```

(EW-30) Naive Bayes classifier using equal-width histograms; the number of bins is fixed to 30 (`-B 30`).

```
java  weka.classifiers.bayes.NaiveBayesParametrized -t iris.arff \
-E "weka.estimators.EqualWidthEstimator -B 30"
```

Appendix E

Program Calls Used for Presentation Methods

E.1 Bin Lists

Bin List Generated Using the MultiTUBE Estimator java -Xmx1500M weka.estimators.MultiTUBE -i musk1-propositional.arff -V17 -B 10 -N ¿musk1-B10-V17.binpos

E.2 Bin Lists for Two-Class Problems

Bin List Generated Using MultiTUBEClusterer

```
java -Xmx1500M weka.clusterers.MultiTUBEClusterer \
-t musk1_propositional.arff -T musk1-propositional.arff  -V 19 -L last -C 1 -X
-E "weka.estimators.MultiTUBE -B 50 -N" >musk1-B50-C1-V19.binlist
```

Bin List After Using the Mixing of Binnings Method

```
java -Xmx1500M weka.clusterers.MultiTUBEClusterer \
-t musk1_propositional.arff -T musk1-propositional.arff  -V 19 -L last -C 3 -Y
-E "weka.estimators.MultiTUBE -B 10 -N" >musk1-B50-C3-Y2-V19.binlist
```

E.3 Bin Position Overview

```
java -Xmx1500M weka.clusterers.MultiTUBEClusterer \
-t musk1_propositional.arff -T musk1-propositional.arff  -V 9 -L last -C 1 -X \
-E "weka.estimators.MultiTUBE -B 10 -N" >musk1-B10-C1-V9.binpos
```

Appendix F

Program Calls Used in Application Clustering

F.1 Generating the Example Datasets

Introduction: WEKA Subspace Data Generator For the evaluation of the application clustering all datasets used were generated using WEKA's subspace data generator (generating arff datasets). The class SubspaceCluster has parameters to set the number of attributes (`-a 4`), if a class attribute is generated or not (`-c`), the percentage of generated noise (`-P 10` for 10 percent noise), the output file name for the generated arff-file (`-o test.arff`) and the value range for all attributes that are not listed in the attribute list of the cluster (`-s -10.0,10.0`).

Several clusters can be defined with each a `-C` parameter list containing: the type of cluster specified (`-G` for Gaussian, `-A` for uniformly distributed), the dimensions of the cluster (`-D ..,..,...` for uniform distributions the minimal and the maximal value of the range, for Gaussian distributions the mean value and the standard deviation are given), and the number of instances in this cluster (`-N 100..200`).

In the example below the WEKA subspace clusterer generates two clusters. The first cluster is normally distributed around the mean point $(2.0, 4.0, .., ..)$ and has the standard deviation 1.0 in both attributes. This cluster is clustered in the subspace of the first two attributes only and is normally distributed in the last two attributes in the range $[-10.0, 10.0]$. A value between 100 and 200 is randomly selected as number of instances in this cluster.

The second cluster is a subspace cluster in the third and fourth attribute

and is there randomly uniform distributed between 8.0 and 9.0 in attribute 3 and between 6.0 and 7.0 in attribute 4. The first two attribute values of the instances in this cluster are uniformly distributed in the range $[-10.0, 10.0]$. For this second cluster 300 instances are generated in this fashion.

```
java weka.datagenerators.clusterers.SubspaceCluster  -P 10 -c -a 4 \
-s -10.0,10.0 -o test.arff \
-C "-G 1,2  -D 2.0,1.0,4.0,1.0 -N 100..200" \
-C "-A 3,4  -D 8.0,9.0,6.0,7.0 -N 300..300"
```

Example 1: Dataset with three clusters in fifteen dimensions, each with three relevant attributes.

```
java weka.datagenerators.clusterers.SubspaceCluster  -P 10 -c -a 15 \
-s -3.4,12.6 -o appcluster_multi_3_Vers2.arff \
-C "-G 1,2,3  -D 2.0,1.0,4.0,1.0,1.0,1.0 -N 200..200" \
-C "-G 1,2,4  -D 8.0,1.0,0.0,1.0,4.0,1.0 -N 200..200"  \
-C "-G 2,3,4  -D 10.0,1.0,7.0,1.0,9.0,1.0 -N 200..200"
```

Example 2: Dataset with three clusters in four dimensions, each with three relevant attributes.

```
java weka.datagenerators.clusterers.SubspaceCluster  -P 10 -c -a 4 \
   -s -2.3,12.6 -o appcluster_multi_3_Vers3.arff \
-C "-G 1,2,3  -D 2.0,1.0,4.0,1.0,1.0,1.0 -N 200..200" \
-C "-G 1,2,4  -D 8.0,1.0,0.0,1.0,4.0,1.0 -N 200..200"  \
-C "-G 2,3,4  -D 10.0,1.0,7.0,1.0,9.0,1.0 -N 200..200"
```

Example 3: Dataset with eight clusters in four dimensions, with three relevant attributes each.

```
java weka.datagenerators.clusterers.SubspaceCluster  -P 10 -c -a 15 \
    -s 0.0,10.0 -o appcluster_multi_3_Vers4.arff \
-C "-G 1,2,3  -D 2.0,1.0,4.0,1.0,8.0,1.0 -N 500..500" \
-C "-G 1,2,4  -D 4.0,1.0,8.0,1.0,4.0,1.0 -N 500..500" \
-C "-G 2,3,4  -D 2.0,1.0,2.0,1.0,8.0,1.0 -N 500..500" \
-C "-G 1,2,3  -D 8.0,1.0,4.0,1.0,4.0,1.0 -N 500..500" \
-C "-G 1,2,4  -D 2.0,1.0,8.0,1.0,2.0,1.0 -N 500..500" \
-C "-G 2,3,4  -D 2.0,1.0,8.0,1.0,2.0,1.0 -N 500..500" \
```

```
-C "-G 1,2,3  -D 4.0,1.0,4.0,1.0,2.0,1.0 -N 500..500" \
-C "-G 1,2,4  -D 8.0,1.0,8.0,1.0,8.0,1.0 -N 500..500"
```

Example 4: Example with diagonally set clusters. 2 datasets.

```
java weka.datagenerators.clusterers.SubspaceCluster  -P 10 -c -a 2 \
     -o appcluster_multi_3_Vers11.arff \
-C "-G 1,2  -D 2.0,1.0,2.0,1.0 -N 200..200" \
-C "-G 1,2  -D 7.0,1.0,2.0,1.0 -N 200..200" \
-C "-G 1,2  -D 12.0,1.0,2.0,1.0 -N 200..200"

java weka.datagenerators.clusterers.SubspaceCluster  -P 10 -c -a 2 \
     -o appcluster_multi_3_Vers12.arff \
-C "-G 1,2  -D 2.0,1.0,2.0,1.0 -N 200..200" \
-C "-G 1,2  -D 7.0,1.0,2.0,1.0 -N 200..200" \
-C "-G 1,2  -D 12.0,1.0,4.5,1.0 -N 200..200" \
-C "-G 1,2  -D 4.5,1.0,7.0,1.0 -N 200..200"
```

Example 5: Datasets with Oblong-shaped Clusters. .

```
java weka.datagenerators.clusterers.SubspaceCluster  -P 10 -c -a 2 \
    -s 0.0,10.0 -o appcluster_multi_3_Vers8.arff \
-C "-A 1,2  -D 2.0,3.0,2.0,3.0  -N 500..500" \
-C "-A 1,2  -D 2.0,3.0,8.0,14.0  -N 500..500"
```

Example 6: Dataset with Non-Convex Cluster.

```
java weka.datagenerators.clusterers.SubspaceCluster  -P 10 -c -a 2 \
-o appcluster_multi_3_Vers9.arff \
-C "-A 1,2  -D 2.0,3.0,1.0,6.0  -N 500..500" \
-C "-A 1,2  -D 3.0,8.0,5.0,6.0  -N 500..500" \
-C "-A 1,2  -D 7.0,8.0,1.0,5.0  -N 500..500" \
-C "-A 1,2  -D 0.0,10.0,-1.0,8.0  -N 100..100"
```

F.2 Clustering with TUBE Clusterer

Introduction: TUBE Clusterer The class weka.clusterers.MultiTUBEClusterer clusters the data given in arff format (`-t test.arff`), ignoring the attribute

given with its index as the `-L` parameter. Before the clustering the multi-dimensional space is discretized using the multidimensional TUBE estimator (`MultiTUBE`). The parameters of the MultiTUBE discretizer can be set via the `-E` parameter.

```
java weka.clusterers.MultiTUBEClusterer -t test.arff -L last \
 -E "weka.estimators.MultiTUBE -B 10"
```

Evaluation of the TUBE Clusterer All tests done to evaluate the TUBE clusterer used the default setting of 100 bins for the discretization step.

```
java weka.clusterers.MultiTUBEClusterer -t test.arff -L last
```

Appendix G

Program Calls Used in Application Multiple-Instance Learning

G.1 Using TUBE Clusters for MI Classification

Introduction: TUBE Multiple-instance Classifier TUBEMIC performes clustering to find a cluster defining the positive concept area. In order to do this it uses D_{diff} (`-Y 2`) instead of the density for the bin 'height'. Bins are only accepted as part of the positive concept cluster if their density is above a certain threshold (`-P 90` sets the threshold to 90 percent). Also, in all examples TUBEMIC uses the mixing of binnings (`-C 3`). The number of bins is set with the parameter list of the MultiTUBE estimator in the `-E` parameter.

(TMIC-90) TUBE multiple-instance classifier using 90 percent as density threshold (`-P 90`); the mixing of binnings (`-C 3`) was performed with 10 bins (`-B 10`) and uses D_{diff} (`-Y 2`) instead of the density for the bin 'height'.

```
java weka.classifiers.mi.TUBEMIC -t eastwest_relational.arff -P 90 \
-C "weka.clusterers.MultiTUBEClusterer -L last -C 3 -Y 2 \
-E \"weka.estimators.MultiTUBE -B 10\" "
```

(TMIC-70) and (TMIC-60) Same as TMIC-90 but with different threshold settings (`-P 70` and `-P 60`).

(MIDD) The results of the TUBE multiple-instance classifier were compared with the Multiple-instance Diverse Density classifier

```
java weka.classifiers.mi.MIDD -t eastwest_relational.arff
```

G.2 Improving the Efficiency of the Diverse Density Algorithm

Introduction: Diverse Density Augmented with TUBE The standard Diverse Density classifier is implemented in the class MIDD. The TUBEDD class implements the Diverse Density (MIDD) classifier augmented with TUBE. The TUBE method computes starting points for the Diverse Density method in two ways: as the centre point of the modes (`default`) bins or as randomly selected training points from the mode bins (`-S 6`).

Mixing of Binnings Method The TUBE augmented DD method uses the mixing of binnings method, which is implemented in the TUBE clusterer's class MultiTUBEClusterer (`-C 3`, and for using D_{diff}: `-Y 2`). The number of bins is given with the specification of a MultiTUBE estimator (`-E "weka.estimators.MultiTUBE -B 5"`).

(TUBE-5) Diverse Density augmented with TUBE: using five bins (`-B 5`) in the histograms generated for the mixing of binnings (`-C 3 -Y 2`) and using the centre point of the mode (`default`) as starting point.

```
java weka.classifiers.mi.TUBEDD -t eastwest_relational.arff \
-C "weka.clusterers.MultiTUBEClusterer -L last -C 3 -Y 2 \
-E \"weka.estimators.MultiTUBE -B 5\" "
```

(TUBE-10) Diverse Density augmented with TUBE: using ten bins (`-B 10`) in the histograms generated for the mixing of binnings (`-C 3 -Y 2`) and using the centre point of the mode (`default`) as starting point.

```
java weka.classifiers.mi.TUBEDD -t eastwest_relational.arff \
-C "weka.clusterers.MultiTUBEClusterer -L last -C 3 -Y 2 \
-E \"weka.estimators.MultiTUBE -B 10\" "
```

(TUBE-10-rand) Diverse Density augmented with TUBE: using ten bins (`-B 10`) in the histograms generated for the mixing of binnings (`-C 3 -Y 2`), and using a training instance randomly selected from the concept area (`-S 6`) as starting point.

```
java weka.classifiers.mi.TUBEDD -t eastwest_relational.arff -S 6 \
-C "weka.clusterers.MultiTUBEClusterer -L last -C 3 -Y 2 \
-E \"weka.estimators.MultiTUBE -B 10\" "
```

(MIDD) Multiple-instance Diverse Density classifier.

```
java weka.classifiers.mi.MIDD -t eastwest_relational.arff
```

Bibliography

[1] Charu C. Aggarwal, Joel L. Wolf, Philip S. Yu, Cecilia Procopiuc, and Jong Soo Park. Fast algorithms for projected clustering. *SIGMOD Rec.*, 28(2):61–72, 1999. 138

[2] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *ACM SIGMOD Conference*, pages 94–105, 1998. 27, 138

[3] Hald Anders. *A history of parametric statistical inference from Bernoulli to Fisher, 1713-1935*. Springer, New York, 2007. 2, 4

[4] Stuart Andrews, Ioannis Tsochantaridis, and Thomas Hofmann. Support vector machines for multiple-instance learning. In *In Advances in Neural Information Processing Systems 15*, pages 561–568. MIT Press, 2003. 155, 157

[5] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: Ordering points to identify the clustering structure. In *Proc. ACM SIGMOD99 Int. Conf. on Management of Data*, pages 49–60, 1999. 138, 139

[6] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007. 57, 61, 76, 81, 115, 118, 147

[7] Stephen D. Bay. Multivariate discretization of continuous variables for set mining. In *In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 315–319. Morgan Kaufmann, 2000. 56

[8] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002. 123, 139

[9] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *CART: Classification and Regression Trees*. Wadsworth International Group, Belmont, California, 1984. 3, 18, 19, 20, 49

[10] Jason Catlett. On changing continuous attributes into ordered discrete attributes. In *Proc. Machine Learning - EWLS-91*, pages 164–178, 1999. 56

[11] Yann Chevaleyre and Jean-Daniel Zucker. A framework for learning rules from multiple instance data. In *Proceedings of the 12th European Conference on Machine Learning (ECML-01)*, pages 49–60. Springer-Verlag, 2001. 155

[12] Scott Davies and Andrew Moore. Interpolating conditional density trees. *A. Darwiche, N. Friedman (Eds.), Uncertainty in Artificial Intelligence*, 18:119–127, 2002. 80

[13] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood estimation from incomplete data via the EM algorithm. *Journal Royal Statistical Society B*, 39:1–38, 1977. 139, 140

[14] Thomas G. Dietterich, Richard H. Lathrop, and Tomas LozanoPerez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997. 153, 155, 156, 159

[15] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 194–202. Morgan Kaufmann, 1995. 55, 78, 80

[16] Robert P.W. Duin. On the choice of smoothing parameters for parzen estimators of probability density functions. *IEEE Transactions on Computers*, 25:1175–1179, 1976. 14

[17] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases. In *Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996. 138

[18] Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous valued attributes for classification learning. In *Proc. of the*

Thirteenth International Joint Conference on Artificial Intelligence, pages 1022–1027, 1993. 25, 32, 37, 42, 56, 80, 180

[19] James Foulds and Eibe Frank. Revisiting multiple-instance learning via embedded instance selection. In *Proc 21st Australasian Joint Conference on Artificial Intelligence*, Auckland, New Zealand. Springer, 2008. 162

[20] Eibe Frank. *Pruning Decision Trees and Lists*. PhD thesis, Department of Computer Science, University of Waikato, 2000. 19, 23, 24

[21] Eibe Frank and Ian H. Witten. Making better use of global discretization. *Proc. of the Sixteenth International Conference on Machine Learning*, pages 115–123, 1999. 53

[22] Eibe Frank and Xin Xu. Applying propositional learning algorithms to multi-instance data. Technical Report 06/03, Department of Computer Science, University of Waikato, 2003. 155

[23] Mark A. Hall. *Correlation-based Feature Selection for Machine Learning. PhD dissertation*. PhD thesis, Department of Computer Science, University of Waikato, 1998. 13

[24] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, second edition, 2006. 2, 22, 40, 54, 57, 135, 136, 139, 140, 150

[25] John A. Hartigan. *Clustering Algorithms*. John Wiley & Sohn, Inc., New York, 1975. 135

[26] Trevor Hastie, Robert Tibishirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2001. 1, 18, 32

[27] David C. Hoaglin, Frederick Mosteller, and John W. Tukey. *Understanding Robust and Exploratory Data Analysis*. Wiley, 1983. 3

[28] Geoffrey Holmes, Bernhard Pfahringer, Richard Kirkby, Eibe Frank, and Mark Hall. Multiclass alternating decision trees. In *Proc 13th European Conference on Machine Learning*, Helsinki, Finland, pages 161–172. Springer, 2002. 20

[29] Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. In *Machine Learning*, pages 63–91, 1993. 56, 80

[30] Jenq-neng Hwang and Shyh-rong Lay. Nonparametric multivariate density estimation: A comparative study. *IEEE Trans. Signal Processing*, 42:2795–2810, 1994. 11

[31] Anil K. Jain, M. Narasimha Murty, and Patrick. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999. 132, 135

[32] Ruoming Jin, Yuri Breitbart, and Chibuike Muoh. Data discretization unification. In *Proceedings of the 7th IEEE International Conference on Data Mining*, pages 183–192, 2007. 26

[33] Karin Kailing, Hans-Peter Kriegel, Alexey Pryakhin, and Matthias Schubert. Clustering multi-represented objects with noise. In *Proc. ACM SIGMOD99 Int. Conf. on Management of Data*, pages 49–60, 1999. 154

[34] Randy Kerber. ChiMerge: Discretization of numeric attributes. In *AAAI-92, Proceedings Ninth National Conference on Artificial Intelligence*, pages 129–134. AAAI Press/The MIT Press, 1992. 55, 56

[35] Donald Ervin Knuth. *The art of computer programming.* Addison Wesley, 1973. 39

[36] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence, Special issue on relevance*, 97:273–324, 1997. 13

[37] Ron Kohavi and Mehran Sahami. Error-based and entropy-based discretization of continuous features. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 495–502. Morgan Kaufmann, 1996. 56

[38] Daphne Koller and Mehran Sahami. Toward optimal feature selection. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 284–292. Morgan Kaufmann, 1996. 13

[39] Ming Li and Paul Vitalnyi. *An introduction to Kolmogorov complexity and its applications.* Springer, New York, 1997. 25

[40] Huan Liu, Farhad Hussain, Chew Lim Tan, and Manoranjan Dash. Discretization: An enabling technique. *Data Min. Knowl. Discov.*, 6(4):393–423, 2002. 54, 55

[41] Philip M. Long and Lei Tan. PAC learning axis-aligned rectangles with respect to product distributions from multiple-instance examples. In *COLT '96: Proceedings of the ninth annual conference on Computational learning theory*, pages 228–234, New York, NY, USA, 1996. ACM. 159

[42] *Machine Learning.* (Monthly Journal), Kluwer Academic Publishers, Boston, 1986-. 1

[43] Oded Maron. *Learning from ambiguity.* PhD thesis, Massachusetts Institute of Technology, 1998. 123, 153, 157, 158, 160, 161, 171

[44] Oded Maron and Tomas Lozano-Perez. A framework for multiple-instance learning. In *Advances in Neural Information Processing Systems*, pages 570–576. MIT Press, 1998. 153, 155, 157

[45] Tom M. Mitchell. The discipline of machine learning, technical report. 1, 182

[46] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, 1997. 18, 19, 22, 27

[47] Claude Nadeau and Yoshua Bengio. Inference for the generalization error. *Machine Learning*, 52:239–281, 2003. 62, 81, 163, 165

[48] Emanuel Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33:1065–1076, 1962. 2, 11

[49] Bernhard Pfahringer. Compression-based discretization of continuous attributes. In *Proceedings of the 12th International Conference on Machine Learning*, pages 456–463. Morgan Kaufmann, 1995. 56

[50] John Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993. 3, 18, 20, 21, 25, 98

[51] Jorma Rissanen. *Information and complexity in statistical modeling.* Springer, New York, 2007. 25

[52] Stuart J. Russell and Peter Norvig. *Artificial Intelligence.* Prentice Hall, 2003. 20

[53] Cullen Schaffer. Selecting a classification method by cross-validation. *Machine Learning*, 13:135–143, 1993. 17, 22, 27

[54] Gabi Schmidberger and Eibe Frank. Unsupervised discretization using tree-based density estimation. In *Proc 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, Porto, Portugal, pages 240–251. Springer, 2005. 29

[55] David W. Scott. On optimal and data-based histograms. *Biometrika*, 66:605–610, 1979. 3, 5

[56] David W. Scott. *Multivariate Density Estimation: Theory, Practice and Visualization*. John Wiley & Sohn, Inc., 1992. 7, 12, 13, 18, 29

[57] David W. Scott and Stephan R. Sain. *Multi-dimensional Density Estimation*, pages 229–263. Elsevier, 2004. 17

[58] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. WaveCluster: A multi-resolution clustering approach for very large spatial databases. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 428–439, 24–27 1998. 139

[59] Sven Siggelkow. *Feature Histograms for Content-Based Image Retrieval*. PhD thesis, Albert-Ludwigs-Universität Freiburg, Fakultät für Angewandte Wissenschaften, Germany, December 2002. 54

[60] Bernard W. Silverman. *Density Estimation*. Chapman and Hall, 1986. 12, 15, 16, 17, 42, 60, 125, 126

[61] Padhraic Smyth. Model selection for probabilistic clustering using cross-validated likelihood. *UCI-ICS Technical Report No.98-09*, 1998. 17

[62] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society (Series B), 36*, 36(2):111–147, 1974. 17

[63] George R. Terrell and David W. Scott. Variable kernel density estimation. *The Annals of Statistics*, 20:1236–1265, 1992. 12

[64] Christopher S. Wallace. *Statistical and inductive inference by minimum message length*. Springer, New York, 2005. 25

[65] Jun Wang and Jean-Daniel Zucker. Solving the multiple-instance problem: A lazy learning approach. In *Proceedings of the 17th International Conference on Machine Learning*, pages 1119–1125. Morgan Kaufmann, 2000. 155

[66] Wei Wang, Jiong Yang, and Richard R. Muntz. STING: A statistical information grid approach to spatial data mining. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *Twenty-Third International Conference on Very Large Data Bases*, pages 186–195, Athens, Greece, 1997. Morgan Kaufmann. 26, 139

[67] Nils Weidmann, Eibe Frank, and Bernhard Pfahringer. A two-level learning method for generalized multi-instance problems. In *Proc 14th European Conference on Machine Learning*, Cavtat-Dubrovnik, Croatia, pages 468–479. Springer, 2003. 156, 159, 160

[68] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2 edition, 2005. 4, 6, 62, 79, 140, 141, 163, 197

[69] Ying Yang and Geoffrey Webb. Proportional k-interval discretization for naive-bayes classifiers. In *Proceedings of the 12th European Conference on Machine Learning*, pages 564–575. Springer, 2001. 55

[70] Ying Yang and Geoffrey I. Webb. A comparative study of discretization methods for naive-bayes classifiers. In *Proceedings of PKAW 2002: The 2002 Pacific Rim Knowledge Acquisition Workshop*, pages 159–173, Tokyo, 2002. 47, 80