Part I

Part I

Chapter 3

Discretization - Framework, Preliminaries and State of the Art

In this chapter, we discuss discretization in general and formulate the discretization problem. We give certain preliminary definitions and explain the notion of optimal discretizations. Then we discuss the taxonomy of discretization methods and without being exhaustive, we briefly describe some known and novel discretization methods that come under these taxonomies. We also discuss the role of discretization as a preprocessing step for decision trees and naive-bayes classifiers and what type of discretization is suited for which classifier. At the end we present our conclusions and discuss future trends related to discretization schemes.

3.1 Introduction

In supervised learning, we come across numeric and symbolic (continuous and discrete) data. Since, most of the methods only deal with symbolic data, it is necessary to transform numeric attributes into symbolic ones. This process is called discretization. It consists of cutting the domain of numeric attribute into a finite number of intervals and representing each interval by a different value known as the discretization interval or discretization point.

Discretization is a general purpose preprocessing method that is used as a preprocessing technique in the data preparation process in KDD. While they are critical in the case of decision tree methods, discretization methods can also be used for bayesian networks, rule-set

algorithms or logistic regression. However, discretization methods have mainly been evaluated using decision trees such as CART [75] or C4.5 [67]. Many real-world classification algorithms are hard to solve unless the continuous attributes are discretized and Kusiak [5] emphasized that the choice of the discretization technique has important consequences on the induction model used. In the Top-Down Induction of Decision Trees family, the algorithms for discretization are based mostly on binarization within a subset of training data [69]. A simple unsupervised discretization procedure divides the range of a continuous variable into equal-width intervals or equal-frequency intervals. While, supervised methods use information quality or statistical quality based measures to determine the interval boundary points. These algorithms reduce the number of attributed values maintaining the relationship between the class and attribute values.

In the top-down induction of decision trees family, a simple unsupervised discretization procedure divides the range of a continuous variable into equal-width intervals or equal-frequency intervals. While, supervised methods use information quality or statistical quality based measures to determine the interval boundary points. Out of these, top-down methods as FUSBIN and MDLPC [30, 121] start with one interval and split intervals in the process of discretization. While, bottom-up methods like FUSINTER[30] and Chi-Merge [103] split completely all the continuous values of the attribute and merge intervals in the process of discretization. Apart from these local discretization methods, global discretization [90], such as binning, produce a mesh over the entire nŋdimensional continuous instance space, where each feature is partitioned into regions independent of the other attributes. In this thesis, most of our focus will be on top-down and bottom-up strategies for providing comparisons in terms of quality and prediction rates [31].

3.1.0.3 Formulation

The discretization of a continuous attribute $X(.)$ consists in splitting the domain of continuous attribute $X(.)$, into k intervals $I_j, j = 1, ..., k$, with $k \geq 1$. Each interval I_j has two end points denoted by $[d_{j-1}, d_j]$ which are called the discretization points. Specifying a partition in intervals means that we only look for ordered partitions over the domain of attribute $X(.)$. To define an ordered partition we consider $x_1, ..., x_n$, a partition in k intervals and we take

an interval $I_l = x_i, ..x_j, .., x_n$. Now, this interval is said to be ordered if and only if for every two elements x_i and x_j of this interval I_l:

$$\forall x_i, x_j \in I_l (x_i \leq x_j \text{ and } x_i \in I_l \text{ and } x_j \in I_l),$$

Thus, we only look for ordered partitions over the domain of $X(.)$.

In the framework of supervised learning, the discretization points of attribute $X(.)$ are determined by taking into account the particular attribute $C(.)$. The purpose of the method is to build a model which can predict class $C(.)$. The purpose of the method is to build a model which predicts class $C(.)$ relative to $X(.)$ [31].

Thus, we look for the discretization points d_j with $j \in 1, ..., k-1$. Then, the attribute $X(.)$ is replaced by an attribute $X^*(.)$ whose values are in the set $1, .., k$. Hence, for all examples ω from the population Ω,

$$If X(\omega) \leq d_1 then X^*(\omega) = 1$$
$$Else If d_{j-1} \leq X(\omega) < d_j then X^*(\omega) = j$$
$$Else If X(\omega) \geq d_k then X^*(\omega) = k$$

Moreover, to predict the attribute $C(.)$, we try to be as close as possible of the following situation:

$$\forall j \in 1, ..., k, \exists c_l \in c_1, ..., c_m; P(c_l/I_j) \approx 1.$$

Ideally, each interval of the discretization should contain only examples belonging to the same class. In other words, two different examples belonging to the same interval of discretization have the same value for $C(.)$. The probabilities $P(c_l/I_j)$ can be estimated from the empirical frequencies $f(c_l/I_j)$ computed on the learning sample set by [31]:

$$f(c_l/I_j) = \frac{card(\omega \in \Omega; X(\omega) \in I_j, C(\omega) = c_l)}{card(\omega \in \Omega; X(\omega) \in I_j)}$$

which corresponds to the frequency of class c_l in the interval I_j.

3.1.1 Definitions

3.1.1.1 Split

Split Ω into Ω_1 and Ω_2 which consists of splitting $X(\Omega)$ in two intervals I_1 and I_2 and setting $\Omega_1 = X^{-1}(I_1)$ and $\Omega_2 = X^{-1}(I_2)$. The cutting point belongs to I_1 but not to I_2.

3.1.1.2 Boundary Points

Let $X(\Omega) = x_1, ..., x_j, x_{j+1}, ..., x_a$ be the ordered set of values of $X(.)$, $x_1 < ... < x_j < x_{j+1} < ... < x_a$. Let Ω_j and Ω_{j+1} be the set of examples of Ω that take values x_j and x_{j+1} respectively:

$$\Omega_j = \omega = \Omega / X(\omega) = x_j \text{ and } \Omega_{j+1} = \omega \in \Omega / X(\omega) = x_{j+1}.$$

Assume d_j is situated between x_j and x_{j+1}, such that there is $\rho =]0, 1[$, such that $d_j = \rho \times x_j + (1 - \rho) \times x_{j+1}$. d_j is called a boundary point if and only if the classes of examples of Ω_j are not all the same as those of the examples of Ω_{j+1}; that is to say,

$$\exists \omega \in \Omega_j \exists \omega^{/} \in \Omega_{j+1} \text{ such that } C(\omega) \neq C(\omega^{/}).$$

Let B be the set of boundary points. We have $b = card(B)$. Fayyad and Irani [121] have conjectured that the discretization points d_j can only be boundary points. They have proved this conjecture using an entropy based criterion that the discretization points d_j can only be the boundary points B.

3.1.1.3 Data Representation

Let us consider a split into k intervals. It is possible to associate a contingency table T_c with m rows and k columns corresponding to classes and intervals respectively. We introduce several definitions:

1. n_{ij} is the size of the data sample whose examples belong to I_j and to the class c_i.
2. $n_{.j}$ is the size of the data sample who example belong to I_j.
3. $n_{i.}$ is the number of examples whose class is c_i.

4. n is the size of the data sample.

With each discretization in k intervals, it is possible to associate a contingency array T_c with m rows and k columns. The rows correspond to the classes and the columns to the intervals.

3.1.1.4 Runs

A runs is a set of points placed between two boundary points. When there is no superimposition, all the points of a run have the same class $C(.)$. Else, the associated run will be reduced to this unique value and will contain a mix of classes. A run is represented by a vector that describes the number of observations corresponding to each class for the underlying interval of run. The run is represented as:

$$R = (R_1, R_2, ..., R_r)$$

Here $r = B + 1$, where B is the number of boundary points.

3.2 Types of Discretization

Because of the fact, that we concentrate on top-down and bottom-up supervised discretization methods, we briefly define these two strategies as defined in [31].

3.2.1 Top Down Methods

All Top-Down methods we know, are based upon a single strategy:

1. All the examples ω are sorted by increasing $X(.)$. This gives a list of $k - 1$ boundary points.

2. From each boundary point, a binary partition is built by splitting the population at this point. The point which optimizes a given criterion is then retained as the cut point. The process stops if no improvement can be done.

3. Step 2 is recursively repeated on the two new sub-populations.

At step 2, the chosen discretization point d_t^* leads to the best binary partition on Ω. Let Ω_1 and Ω_2 be the two elements of this binary partition and $m_j = card(C(\Omega_j))$ the number of different classes in the sub-sample Ω_j.

Moreover, at step 2, both a criterion and a stopping rule must be specified. We present one of the several used criteria.

3.2.2 Bottom-Up Methods

Most bottom-up methods are based upon following strategies:

1. The examples are sorted according to increasing value of $X(.)$.
2. Each attribute value $x_j \in X(\Omega)$ forms the center of an interval I_j whose bounds d_j separate two consecutive values of $X(\Omega)$, so that $d_j = \frac{x_{j-1}+x_j}{2}$.
3. To each interval I_j is associated a distribution T_j where n_{ij} stands for the number of examples belonging to I_j and having class c_i. We compute the value of a criteria associated to the matrix formed by the juxtaposition of the two columns T_j and $T_j + 1$ corresponding to the two adjacent intervals.
4. We merge the adjacent intervals I_q and I_{q+1} with the best criteria value.
5. The process stops when no merging is possible. Else, we return to step 4 with k-1 intervals.

3.3 Optimal Discretization

In this section, we discuss the notion of optimal discretization and discuss the possibilities of building optimal partitions.

3.3.1 Measuring the Quality of a Partition

The best quality discretization is actually based on finding the partitions from which we can predict the class $C(.)$ best. We necessarily have to compare partitions containing different number of intervals. The quality measure should take into account the increase in complexity

induced by an excessive partitioning. Actually, a very thin partitioning, with, for instance, one example in each interval, will have an error rate by resubstitution nearly equal to zero on the learning sample. It will surely fail, however in generalization and also will have very high variance as discussed before.

Thus, when the complexity or number of partitions are taken into account while evaluating a partition this leaves out measures based on the estimation of the probabilities exclusively from the sole empirical frequencies. For instance, the measure of purity such as those proposed by Brieman [75], or others like information gain, have the particularity to favor excessive partitioning of the domain of X(.) [44, 100]. The goal is to predict the attribute $C(.)$ with a minimum error rate.

There are several ways to judge the relevance of partitioning [31]:

- Using the minimum descriptor length principle [121]: The minimum descriptor length of describing the class of all the data examples is compared to the descriptor length of the partition to be evaluated. Thus, if there is second length is lower the partition is accepted.

- By measuring the association between intervals defined over the continuous attribute and the class to predict using a confidence measure of the χ^2 type (e.g. Tshuprow [1] , Cramer [44]). Thus, it is possible, on one hand, to evaluate the reliability of a discretization and, on the other hand, to compare the quality of partitions with different number of intervals.

- Using a criterion based on bayesian theory [79] enabling the selection of the discretizations that optimize that criterion. The complexity criterion is introduced by a choice of the distribution of prior probabilities of the intervals.

- Adapting the notion of information gain by correcting its tendency to favor over-partitioning. A measure which takes into account the size of the data sets has been introduced to find the optimal partition [29]. The information gain introduced by a new split is balanced by the decrease in the numbers of examples of the subgroup in order to estimate the conditional probabilities of belonging to a class relative to defined intervals.

3.3.2 Framework

This section introduces Fisher's algorithm [122] and the point of view of Lechevallier [135] and Zighed [31]. This algorithm is based on two fundamental properties:

- Ordering element property as defined in subsection formulation.
- Additivity of the quality measure: Let $(I_1, ..., I_k)$ be a split in k intervals of $X(\Omega)$. The measure φ is said to be additive if an only if it verifies $\varphi(I_1, ..., I_k) = \sum_{j=1}^{k} \varphi(I_j)$. As a consequence of this property, we have the following property for an additivity measure: If a partition $(x_1, ..., x_i,, I_2, ..., I_k)$ in k intervals is optimal, then the partition $(I_2, ..., I_k)$ in an optimal partition in $k-1$ intervals of the set $x_{i+1}, ..., x_n$.

These properties are proved in [30] for the FUSINTER measure. The second property is basically responsible for the complexity of heuristics presented earlier. Indeed, if the criterion does not satisfy this property, then the worst-case complexity of the bottom-up or top-down methods is at least quadratic. On the contrary, the complexity becomes linear when the measure is additive. Fisher's algorithm has a quadratic complexity which is higher than that of the previous heuristics based on an additive criteria.

3.3.3 An Extension of Fisher's Algorithm

Fischer's algorithm [122] is a dynamic programming procedure. This algorithm has been described like this by Zighed et al in [31] whose idea is to find some relationships between the optimal partition in k intervals of the initial sample set and the optimal partitions in $k-1$ intervals of subsets of this set. It uses the order property restrict the number of possible partitions. The additivity of the measure φ is used to obtain a recurrent equations between optimal partitions. Here, we present an extension of Fisher's and Lechevallier's algorithms by considering the partitioning of a set of runs instead of points. This is the consequence of work of Fayyad and Irani [121] who proved that a run can never be split in an optimal discretization.

Let us consider a set of runs $R_i, 1 \leq i \leq r$. We search an ordered partition that is optimal for the measure φ. We denote by ζ_k^1 this partition, with k being the number of intervals and 1 the first run taken into account. We then have,

$$\zeta_k^1 = R_1, ..., R_{j_1}, R_{j_1+1}, ..., R_{j_2}, ..., R_{j_{k-1}}, ..., R_r$$

Because φ is additive, the value of φ on the previous partition is given by $\sum_{i=1}^{k} \varphi(R_{j_{i-1}+1}, ..., R_{j_2})$, where $j_o = 0$ and $j_k = r$ for the sake of simplicity. This additivity of φ also implies that,

$$\zeta_{k-1}^{j_1+1} = R_{j_1+1}, ..., R_{j_2}, ..., R_{j_{k-1}}, ..., R_r$$

is an optimal partition in $k-1$ intervals of the set of runs $R_i, j_1 + 1 \leq i \leq r$. The problem then consists of finding the first cut point j_1. This point is one of the integers of the interval $[1, r-k+1]$. The optimal partition ζ_k^1 can then be obtained through a minimization process,

$$\varphi(\zeta_k^1) = min_{1 \leq j_1 \leq r-k+1} \varphi(R_1, ..., R_{j_1}) + \varphi(\zeta_{k-1}^{j_1+1})$$

The various partitions $\zeta_{k-1}^{j_1+1}$ are also optimal partitions, so they can be determined with the same procedure. This gives us a series of optimal partitions with a decreasing number of intervals. Thus, we iteratively decrease the value of k by considering the set of partitions ζ_1^l, which are optimal for the set of runs $R_i, l \leq i \leq r$. They are the only partitions to be considered because all the others can be deduced from them by iteratively applying the minimization procedure. But, we have $\zeta_1^l = R_l, ..., R_r$. Thus, the algorithm proceeds as follows:

1. Compute all the partitions ζ_1^l for $1 \leq l \leq r$.

2. For all p, $2 \leq p \leq k$; compute the partitions ζ_p^q for each q of the interval $[1, r-p+1]$:

   - Compute the summations $\varphi(R_q, ..., R_o) + \varphi(\zeta_{p-1}^o)$ for $q \leq o \leq r$.
   - $\varphi(\zeta_p^q)$ is the minimum value of the previous ones.

   At step$k$ of this algorithm, the optimal partition ζ_k^1 is determined when $q = 1$.

3. The optimal partition ζ^* among all the previous optimal partitions is given by $\varphi(\zeta^*) = min_{j=1}^{r} \varphi(\zeta_j^1)$.

3.4 Quality of Discretization

We use three factors to analyze the quality of the obtained discretization.

1. Prediction Accuracy: The goal of the discretization is to make the class $C(.)$ predictable by an attribute $X(.)$. To measure this prediction rate Zighed et al [31] define a notion of prediction accuracy of the achieved discretization as follows:

   The discretization of the attribute X_j from a sample Ω_s, provides k intervals denoted $I_i^j; i = 1, .., k$. For each ω taken from the test sample Ω_t we denote I_i^j the interval to which it belongs after discretization of the sample Ω_s. The point ω will be labeled $C(\omega) = c^*$ if the majority of the points in the Ω_t that are in I_i^j have the class c^*. This corresponds to a bayesian decision rule with a matrix of symmetrical costs and prior probabilities of the classes estimated by the proportion of the individuals belonging to each class in the Ω_t. We measure the quality of the discretization by the rate of good predictions:

   $$\tau_j = \frac{card\{\omega \in \Omega_t / \hat{C}(\omega) = C(\omega)\}}{card\{\Omega_t\}}$$

   We denote by τ_j the good prediction rate resulting from the discretization of X_j obtained by applying a method on the sample Ω_s.

2. Complexity: In measuring the complexity of a discretization we take into account the number of intervals I_{number} obtained in the resulting discretization. Large number of intervals increase the complexity of the induction algorithm that use the discretized input and also because a large number of intervals are likely to increase the discretization bias and variance. Higher discretization variance effects the quality of discretization even if the prediction accuracy is high. This property is explained by Yang et al [140].

3. Robustness: We introduce a concept of robustness defined in [81]. This is equal to the prediction accuracy in terms of the training sample divided by the predication accuracy of the whole population (which is known in our experiments). This measures the degree of accurate estimation of the population from a small training sample.

3.5 State of the Art: Taxonomy of Discretization Methods

In the following sections we present some of the known and recent discretization methods and their types.

Various discretization methods have been proposed. Diverse taxonomies exist in literature to categorize discretization methods. These taxonomies are complementary, each relating to a different dimension along which discretization methods may differ. In [140] the authors divide the discretization methods into either primary or composite. Primary methods accomplish discretization without reference to any other discretization method. Composite methods are built on top of some primary method(s). Below we take the same division of various discretization methods.

3.6 Primary Methods

By in large these primary methods can be classified or distinguished according to the taxonomies:

3.6.1 Contextual and Non-Contextual [31] or Supervised and Unsupervised [59]

In the case of non-contextual or unsupervised methods, we take into account only the continuous attributes and have no use for the class information. Thus, these methods look for the distribution of values of the attributes and in doing so look for the best partition. Some known unsupervised methods divided the range equally into k intervals where k in chosen by the user. However, other methods interact with the attributes in a more complex manner.

Contextual or supervised methods take into account the attributes as well as the classes. These methods refer to the class information when selecting discretization cut points. Supervised methods usually use various criteria which can be entropy based or can use other statistical criteria like chi square to assess the relationship between, intervals and class. However, error-based supervised methods apply a learner to the transformed data and select the intervals that minimize error on the training data.

3.6.2 Univariate and Multivariate [105]

Univariate methods discretize an attribute without reference to attributes other than the class. In contrast, multivariate methods consider relationships among attributes during discretization.

We believe that the above classification of discretization methods broadly distinguished almost all the existing discretization methods, however, other authors [45, 140] have given other classifications which we briefly describe below:

1. Hierarchical discretization utilizes an incremental but conceptually recursive process to select cut points. Hierarchical discretization can be further characterized as either top-down or bottom-up. Top-Down discretization starts with a single interval that encompasses the entire value range, then repeatedly splits it into sub-intervals until some stopping criterion is satisfied. Bottom-up discretization starts with each value in a separate interval, then repeatedly merges adjacent intervals until a stopping criterion is met. It is possible to combine both split and merge techniques. For example, initial intervals may be formed by splitting. A merge process is then applied to post-process these initial intervals.

2. Non-hierarchical discretization creates intervals without forming a hierarchy, i.e forming the intervals sequentially in a single scan through the data.

3. Global methods [59, 140] Global techniques are simple and efficient, as one discretization process is used throughout the entire classification task.

4. Local methods [59, 140] are context dependent and allow different mapping functions for a single attribute in different classification contexts. Thus, these techniques are expensive but may result in the discovery of more useful cut points.

5. Static or Eager Methods [49] perform discretization prior to classification time. This accounts for the vast majority of the discretization methods.

6. Lazy methods [49] generate the mapping function or discretize as it is needed during classification time.

7. Fuzzy discretization [48, 140] creates fuzzy or overlapping partitions. A value may belong to multiple intervals, each with varying degrees of strength.

8. Non-fuzzy discretizations are the vast majority of discretizations as they form exact cut points with no overlapping.

In the following sections without being exhaustive we briefly describe some of the discretization methods falling into the above taxonomies.

3.7 Supervised Hierarchical Discretizations (Univariate)

Among the hierarchical discretization methods, we start by presenting some criteria for top-down and bottom-up discretizations which are most commonly used and both the processes are discussed in chapter 3. We also point out that these are the methods that we used for comparisons during experimentations. These criteria are either entropy or statistically based and are present below:

3.7.1 Top-Down Discretizations

3.7.1.1 χ^2 Statistical Law [103]

$$\chi^2 = \sum_{i=1}^{m} \sum_{j=1}^{2} \frac{(n_{ij} - n_{i.} n_{.j})^2}{n_{i.} n_{.j}}$$

The chosen discretization point, d_t^*, verifies $Gain(d_t^*) = max_t Gain(d_t)$. As a stopping rule, we use a comparison with the theoretical χ^2 value. If $\chi^2(d_t) \leq \chi^2_{(\alpha,m-1)}$ then no improvement is possible.

3.7.1.2 FUSBIN [31]

FUSBIN uses an uncertainty measure which is sensitive to the sample size. These measures are similar to the previous ones but take into account the sample size:

$$h(\Omega) = \sum_{i=1}^{m} \frac{n_{i.}+\lambda}{n+m\lambda} \left(1 - \frac{n_{i.}+\lambda}{n+m\lambda}\right)$$

$$h(\Omega_j) = \sum_{i=1}^{m} \frac{n_{ij}+\lambda}{n_{.j}+m\lambda} \left(1 - \frac{n_{ij}+\lambda}{n_{.j}+m\lambda}\right)$$

$$Gain(d_t) = h(\Omega) - \sum_{j=1}^{2} \frac{n_{.j}}{n} h(\Omega_j)$$

The chosen discretization point, d_t^*, verifies $\text{Gain}(d_t^*) = \max_t \text{Gain}(d_t)$. The process stops when $Gain(d_t^*) \leq 0$. The value of λ is strictly positive and can be computed with the procedure described in [1].

3.7.1.3 MDLPC Criterion

Fayyad and Irani [121] use an informational criterion called Minimum Description Length Principle Cut. It is based upon the information gain as below. The MDLPC criterion consists in looking for points d_t that satisfy.

$$Gain(d_t) > \frac{log_2(n-1)}{n} + \frac{\delta(d_t)}{n}$$

Where, with $m_j = card(C(\Omega_j))$

$$\delta(d_t) = log_2(3^m - 2) - mh(\Omega) + \sum_{j=1}^{2} m_j h(\Omega_j)$$

And by choosing the one which maximizes the $Gain(d_t)$.The process stops if $Gain(d_t)$ is always less or equal to $\frac{log_2(n-1)}{n} + \frac{\delta(d_t)}{n}$

3.7.1.4 CONTRAST Criterion

Van de Merckt [116] propose a criterion that takes into account the homogeneity of the classes and also the point density. Each boundary point d_t is used as the cutting point

$$Cont(d_t) = -\frac{n_{.1}n_{.2}(g_1 - g_2)^2}{\sum_{i=1}^{m}\sum_{j=1}^{2} n_{ij} log_2 \frac{n_{ij}}{n_{.j}}}$$

Where,

$$g_j = \frac{1}{n_{.j}} \sum_{\omega \in \Omega_j}$$

Is the vector of the average values of the sub-sample Ω_j.The chosen point, d_t ,verifies: $Cont(d_t^*) = min_t(Cont(d_t))$. We add to the original CONSTRAST method following stopping rule: $Gain(d_t) \leq 0$.

The exist also other criteria including association measure or attributes dependency measure, such as the t of Tshuprow [1] or the V of Cramer [44].

3.7.2 Bottom-Up Methods

3.7.2.1 The Chi-Merge Method

This method was proposed by Kerber [103] and relies on the use of the statistical criterion. The algorithm was quite simple and can be summed up as follows:

1. The examples are sorted according to increasing value of $X(.)$.
2. Each attribute value $x_j \in X(\Omega)$ forms the center of an interval I_j whose bounds d_j separate two consecutive values of $X(\Omega)$, so that $d_j = \frac{x_{j-1}+x_j}{2}$.
3. To each interval I_j is associated a distribution T_j where n_{ij} stands for the number of examples belonging to I_j and having class c_i. We compute the value of X^2 associated to the matrix formed by the juxtaposition of the two columns T_j and T_j+1 corresponding to the two adjacent intervals.
4. We merge the adjacent intervals I_q and I_{q+1} with the smallest X^2 value when

$$X^2(T_q, T_{q+1}) < X^2(\alpha, m-1)$$

   where, $X^2(\alpha, m-1)$ is read on the X^2 table at the threshold α for $m-1$ freedom degrees.
5. The process stops when no merging is possible. Else, we return to step 4 with k-1 intervals.

3.7.2.2 FUSINTER [30]

Let φ be the chosen criterion. The algorithm develops as follows.

1. All examples are sorted according to increasing values of $X(.)$
2. The set of points gives the first discretization in k intervals. The discretization points are thus, necessarily boundary points. We deduce a matrix T of m rows and k columns.
3. We seek for two adjacent intervals that would most improve the value of the criterion when merged, ie we seek j such that:

$$\varphi(T) - \varphi(..., T_j + T_{j+1}, ...) = max_{i=1}^{k-1}(\varphi(T) - \varphi(..., T_i + T_{i+1}, ...))$$

4. If

$$\varphi(T) - \varphi(..., T_j + T_{j+1}, ...) > 0$$

   Then the two intervals I_j and I_{j+1} are merged.

5. The process is repeated from step 3 with k-1 intervals until no more merging is done at step 4 or until k reaches the value 1.

   If the process stops with k = 1, it means that the discretization of $X(.)$ is of no interest for the determination of $C(.)$

The FUSINTER criterion is built on uncertainty measures and is sensitive to sample size. The generic form of the φ function considered in [31].

$$\varphi(t) = \sum_{j=1}^{k} \alpha H_j(h, \lambda) + (1 - \alpha)\frac{m\lambda}{n_{.j}}$$

where $H_j(h, \lambda)$ is an uncertainty measure derived from a quadratic entropy h by:

$$H_j(h, \lambda) = \frac{n_{.j}}{n} h\,(quadratic - entropy)$$

$$h\,(quadratic - entropy) = \sum_{i=1}^{m} \frac{n_{ij}+\lambda}{n_{.j}+m\lambda}\left(1 - \frac{n_{ij}+\lambda}{n_{.j}+m\lambda}\right)$$

3.7.2.3 MODL

MODL discretization [81] uses bottom-up discretization and aims to optimize naive-bayes classifier. It uses a standard discretization model distributed according to a three-stage prior which is Bayes optimal for a given set of instances to discretize if the value of the following criterion is minimal:

$$log\,(n) + log\binom{n+k-1}{k-1} + \sum_{i=1}^{k} log\binom{n_i+m-1}{m-1} + \sum_{i=1}^{k} (n_i!/n_{i,1}!n_{i,2}!...n_{i,m}!)$$

3.7.3 Other Entropy based Hierarchical Methods

A number of entropyŋbased hierarchical methods exist on discretization. Chiu et al [34] have proposed a hierarchical discretization method based on maximizing the Shannon entropy over the discretized space. This method uses a hillŋclimbing search to find a suitable initial partition of the continuous space into k bins along each axis and then further refines this method to particular intervals to obtain finer intervals.

Catlett's D-2 discretizer [58] makes use of entropyŋbased discretization in decision tree domains as a means of achieving a significant increase in the speed of induction on very large data sets with many continuous features. It uses several stopping criteria such as a minimum threshold for number of samples in one partition, a maximum threshold for the number of partitions, and a minimum threshold for information gain.

Pfahringer [11] uses entropy to select a large number of candidate splitŋpoints and employs a bestŋfirst search with a Minimum Description Length heuristic to determine a good discretization.

3.8 Supervised Non-Hierarchical Discretizations

Most of the supervised discretizations follow a hierarchical strategy but there are also those that don't. One of those strategies is the 1R discretizer explained below:

3.8.1 Holte's 1R Discretizer

Holte [101] presents a simple example of a supervised discretization method. This method, referred to here as 1RD (OneŋRule Discretizer) is a greedy method that first sorts the values of a continuous attribute X and attempts to divide the domain of the attribute into bins that each contain only instances of one particular class. In order to avoid the possibility of forming too many bins, the algorithm is constrained to forms bins of at least some minimum size. In [101] the authors suggests a minimum bin size of 6 based on an empirical analysis of 1R on a number of classification tasks. Thus, the algorithm proceeds after fixing the minimum bin size, in a way such that each discretization interval is made dominant with a single class as much as possible. This is done by selecting discretizationŋpoints such that

moving a partition boundary to add a new value to a particular bin cannot make the count of the dominant class in that bin greater.

3.9 Unsupervised Discretizations

The unsupervised discretization can be grasped as a problem of sorting and separating intermingled probability laws [72]. Mostly, these methods are limited in their application in data mining due too strong statistical hypothesis. However, the two most popular unsupervised methods are explained below:

3.9.1 Equal Width Discretization

This method [58, 103, 59] divides the values of the attribute X between x_{min} and x_{max} into k intervals of equal width. Thus the intervals have width $w = (x_{max} - x_{min})/k$ and the obtained discretization points are at $x_{min} + w, x_{min} + 2w, ..., x_{min} + (k-1)w$. k is a user predefined parameter.

3.9.2 Equal Frequency Discretization

In this method [58, 103, 59] the sorted values of attribute X are divided into k intervals so that each interval contains approximately the same number of training instances. Thus each interval contains $n = k$ adjacent values. k is a user predefined parameter. There are some variations of this method including by authors in [90] who adjust the boundaries to decrease entropy at each interval .

Both the above unsupervised methods potentially suffer much attribute information loss since k is by the user and not determined in line with the properties of the training data. But although they may be deemed simplistic, both methods are often used and work surprisingly well for naive-Bayes classifiers. One reason suggested by [140] is that discretization usually assumes that discretized attributes have Dirichlet priors, and ŚPerfect AggregationŠ of Dirichlets can ensure that naive-Bayes with discretization appropriately approximates the distribution of a numeric attribute.

3.10 Fuzzy Discretizations

We present one fuzzy discretization scheme here. However, other fuzzy discretizations are described in detail in part 2 of this thesis.

3.10.1 Fuzzy Discretization

Fuzzy discretization was proposed by Kononenko for naive-Bayes classifiers [55]. It estimates discretization points trying to achieve the best $p(d_j < X_j \leq d_(j+1)|C = c)$. It initially forms k equal-width intervals $(d_j, d_(j+1)](1 \leq j \leq k)$ using equal width discretization which as discussed above is an unsupervised method. It then estimates $p(d_j < X_j \leq d_(j+1)|C = c)$ from all the training set such that:

$$p(d_j < X_j \leq d_(j+1)|C = c) \approx \frac{\sum_{i=1}^{n} P(\mu,\sigma,i)}{p(C=c)}$$
$$\text{and } P(\mu,\sigma,j) = \int_{d_(j+1)}^{d_j} \frac{1}{\sigma\sqrt{2\Pi}} e^{-1/2(\frac{x-\mu}{\sigma})^2} dx$$

Where, σ is a parameter used to control the fuzzy region of the interval bounds, μ is the mean value and n is the number of examples. The idea behind fuzzy discretization is that small variation of the value of a continuous attribute should have small effects on the attributeŠs probabilities, whereas under non-fuzzy discretization, a slight difference between two values, one above and one below the cut point can have drastic effects on the estimated probabilities. But when the training instancesŠ influence on each interval does not follow the normal distribution, FDŠs performance can degrade [140].

3.11 Dynamic Discretizations

3.11.1 Lazy Discretization (LD)

LD [145] estimates $p(X_i = x_i|C = c)$ for each attribute of each test example. Thus, this method is a lazy method because it delays discretization until a test example actually arrives during classification. Thus, in choosing the value of the attribute X from the test example, it selects a pair of discretization points such that the value is in the middle of its corresponding interval.

As expected, LD has high memory and computational requirements because of its lazy methodology. However, static approaches carry out discretization prior to the learning algorithm starts its process.

3.12 Local Methods

While, all the above described methods are global methods, in this section we consider two local discretization methods.

3.12.1 Histogram based Discretization

This method by Perner and Trautzsch [98] calculates the distribution $p(X|X \in C)p(C)$ of attribute X according to class C. The curve of the distribution is approximated by a first order polynom and the minimum square error method is used for calculating the coefficients:

$$E = \sum_{i=1}^{n} (a_1 x_i + a_0 - y_i)^2$$
$$\text{and } a_1 = \frac{\sum_{i=1}^{n} x_i . i}{\sum_{i=1}^{n} i^2}$$

Where a_0 is either the starting point of this interval or the last point of the preceding interval. The cut points are selected by finding the maximum peak point of the classes situated next to each other.

3.12.2 Evolutionary Discretization

This method by Kwedlo and Kretowski [123] was implemented in EDRL-MD, an evolutionary-algorithm-based system for learning decision rules. EDRL-MD simultaneously searches for the cut points for all quantitative attributes during the generation of decision rules. An evolutionary algorithm (EA) [142] is used. The success of EA is attributed to the ability to avoid local optima, which is its main advantage over greedy search methods.

3.13 Multivariate Discretization

3.13.1 Unsupervised

In a multivariate discretization, each attribute is cut relative to the other attributes of the data set, thus this approach can then provide a global view of the problem and thus, some interesting improvements when univariate discretization methods do not yield satisfactory results. The multivariate unsupervised discretizations normally use clustering techniques to merge the values using all attributes globally. We list some unsupervised multivariate methods below:

1. An approach in [82] performs unsupervised multivariate discretization. For starters this method performs univariate discretization of all attributes. Then further purification is perform by cutting a given attribute based on the supervised discretization of all other attributes previously discretized. Each attribute plays the role of a class attribute one time after another. Finally, the smallest intervals are detected and merged.

2. In [90] the authors form each cluster and consider it as a representation of a particular class and improve the discretization quality by using contextual discretization methods

3. In [105] the discretization process consists of merging the bordering intervals in which the data distribution is the same

However, these techniques give less satisfactory results as compared to methods where class values are inherently dealt with such as the methods below.

3.13.2 Supervised

As said above that when the learning problem is inherently supervised and there exist interactions between the continuous attributes, the methods previously seen will not give satisfactory results. Among these methods we discuss the Hyper-Cluster Finder method [39].

This method is based on forming clusters of same class instances that are closed in the representation space. Thus, the process starts by building a neighborhood graph by using

all predictive attributes to determine which examples are close to others. Next, the edges connecting two instances belonging to different classes cut on the graph to form clusters. Finally, the minimum and maximum values of each relevant cluster are used as discretization points on each predictive attribute. This method focuses on the purity relative to all the attributes even if on a single attribute this is not so. Thus, it is the combination of all predictive attribute intervals that will provide well separated or classified regions in the representation space.

3.14 Composite Discretization Methods

A composite method first chooses some primary discretization method to form the initial intervals. It then focuses on how to adjust these initial intervals to achieve certain goals.

3.14.1 Iterative Discretization

This method [88] initially forms a set of intervals using the unsupervised equal width discretization method. Then it performs the task of refinement of these intervals by iteratively adjusting in order to minimize the naive-Bayes classification error.

It defines two operators: binary merge and split in the middle of contiguous values in that interval. Thus, at each iteration, it makes a pass for each continuous attribute, and applies all split and merge procedures in all possible ways to initial intervals and estimates the classification error of each operation using leave-one-out cross-validation. The refinement using merge or split that yields the lowest error is retained. The process stops when no refinement further reduces the error. This method is computationally expensive as the training set becomes large because of the high number of possible merges and splits.

3.15 Discretization for Specific Classifiers

Many discretization techniques have been developed primarily in the context of a specific type of learning algorithm, such as decision tree learning, decision rule learning, naive-Bayes learning, Bayes network learning, clustering, and association learning. Different types of learning have different characteristics and hence require different strategies of discretization.

For example, decision tree learners can suffer from the fragmentation problem [140]. If an attribute has many values, a split on this attribute will result in many branches, each of which receives relatively few training instances, making it difficult to select appropriate subsequent tests. Hence they may benefit more than other learners from discretization that results in few intervals. Decision rule learners may require pure intervals (containing instances dominated by a single class), while probabilistic learners such as naive-Bayes do not. The relations between attributes are key themes for association learning, and hence multivariate discretization that can capture the inter-dependencies among attributes is desirable.

In order to facilitate understanding this issue, we contrast discretization strategies in two popular learning contexts, decision tree learning and naive-Bayes learning. Although both are commonly used for data mining applications, they have very different inductive biases and learning mechanisms. As a result, they desire different discretization methodologies.

3.15.1 Discretization in Decision Tree Learning

Algorithms such as ID3 [65] and its successor C4.5 [67] are well known exemplars for decision trees (DT). The quadratic time complexity of the popular C4.5 decision tree learning algorithm for non-numeric data increases by a logarithmic factor when numerical attributes are processed. Many practical decision trees including C4.5, use bipartitioning techniques that partition the domain into two best halves. However, considering multisplits of the domain with more than two intervals are more popular these days. In general, obtained results using discrete features are usually more compact, shorter and more accurate than using continuous ones, hence the results can be more closely examined, compared, used and reused. Supervised discretization techniques might reasonably be expected to lead to more accurate classification trees since the partitions they produce are directly related to the class to be predicted. On the other hand one might expect most of the unsupervised techniques to be considerably faster since they involve little more than sorting the data, an operation which is common to all discretization methods.

Dougherty et al. [59] carried out a comparative study of five discretization procedures using 16 data sets from the UC Irvine ML Database Repository. Somewhat surprisingly they found only small differences between the classification accuracies achieved by C4.5 against

other discretization methods. None produced the highest accuracy for all data sets. In particular the local supervised method, C4.5, showed no advantage over other supervised methods: Fayyad and IraniŠs method (MDLPC) [121] achieved the best overall results.

MDLPC has been one of the most popular discretization mechanisms for decision tree learning over years. However, it is not necessarily appropriate for other learning mechanisms such as naive-Bayes learning, which involves all the attributes simultaneously [140]. Furthermore, MDLPC uses the minimum description length criterion (MDL) as its termination condition that decides when to stop further partitioning a quantitative attribute's value range. As An and Cercone [2] indicate, this criterion has an effect to form qualitative attributes with few values. This effect is desirable for decision tree learning, since it helps avoid the fragmentation problem by minimizing the number of values of an attribute. If an attribute has many values, forming a split on those values fragments that data into small subsets with respect to which it is difficult to perform further learning [66]. However, this effect of minimization is not so welcome to naive-Bayes learning because it brings adverse impact.

3.15.2 Discretization in Naive-Bayes Learning

When classifying an example, naive-bayes learning applies Bayes' theorem to calculate the probability of each class given this example. The most probable class is chosen as the class of this example. In order to simplify the calculation, an attribute independence assumption is made, assuming attributes conditionally independent of each other given the class. Although this assumption is often violated in real-world applications, naive-Bayes learning still achieves surprisingly good classification performance. In [95] suggested one reason is that the classification estimation under zero-one loss is only a function of the sign of the probability estimation. The classification accuracy can remain high even while the assumption violation causes poor probability estimation, so long as the highest estimate relates to the correct class. Because they are simple, effective, efficient, robust to noise, and support incremental training, naive-bayes classifiers have been employed in numerous classification tasks.

Appropriate discretization mechanisms for naive-Bayes learning include fixed-frequency

discretization [140], proportional discretization [140] and non-disjoint discretization [145]. For example, when discretizing a quantitative attribute, fixed-frequency discretization (FFD) predefines a sufficient interval frequency k. It then discretizes the sorted values into intervals so that each interval has approximately the same number k of training instances with adjacent (possibly identical) values. By this means, FFD fixes an interval frequency that is not arbitrary but can ensure that each interval contains sufficient instances to allow reasonable probability estimates.

However, FFD may result in inferior performance for decision tree learning. FFD first ensures that each interval contains sufficient instances for estimating the naive-Bayes probabilities. On top of that, FFD tries to maximize the number of discretized intervals to reduce discretization bias. If employed in decision tree learning, this maximization effect of FFD tends to cause a severe fragmentation problem.

The other way around, MDLPC is effective for decision tree learning but not for naive-Bayes learning. Because of its attribute independence assumption, naive-Bayes learning is not subject to the fragmentation problem. MDLPCŠs tendency to minimize the number of intervals has a strong potential to reduce the classification variance but increase the classification bias. As the data size becomes large, it is very likely that the loss through bias increase will soon overshadow the gain through variance reduction, resulting in inferior learning performance [140]. This impact is particularly undesirable, since, due to its efficiency, naive-Bayes learning is very popular for learning from large data. Hence, MDLPC is not a desirable approach for discretization in naive-Bayes learning.

Below we present a discretization method specifically aimed at improving the performance of Naive-Bayes classifiers.

3.15.2.1 Proportional k-Interval Discretization

This is a global, unsupervised and non-parametric method and is presented by Ying Yang [140]. It adjusts discretization bias and variance by adjusting the interval size and the interval number, and further adjusts the naive-BayesŠ probability estimation bias and variance to achieve lower classification error. As mentioned before that the discretization bias and variance are dependent heavily on the interval size and number of intervals. The lower the

number of intervals, the lower the variance but the higher the bias and vice versa. Lower learning error can be achieved by tuning the interval size and number to find a good trade-off between the bias and variance.

This method calculates the number of intervals and frequencies with a very simple formula, $s \times t = n$, where t is the number of intervals, s is the frequency in each interval and n the number of training instances. This method was created in the context of naive Bayes. The author argues that, by setting the number of intervals and frequency proportional to the training data, it reduces the bias and variance. This is also a more sophisticated version of bins.

3.15.3 Other Discretization Methods

Adaptive Quantizers [14] is a method combining supervised and unsupervised discretization. It begins with a binary equal width interval partitioning of the continuous feature. A set of classification rules are then induced on the discretized data (using an ID3ŋlike algorithm) and tested for accuracy in predicting discretized outputs. The interval that has the lowest prediction accuracy is then split into two partitions of equal width and the induction and evaluation processes are repeated until some performance criteria is obtained. While this method does appear to overcome some of the limitations of unsupervised binning, it has a high computational cost as the rule induction process must be repeated numerous times. Furthermore, the method makes an implicit assumption that high accuracy can be attained. For example, on random data, the system might make many splits and a postŋprocessing step needs to be added.

Bridging the gap between supervised and unsupervised methods for discretization, Van de Merckt [116] developed two methods under the general heading of Monothetic Contrast Criterions (MCC). The first criterion, dubbed unsupervised by the author, makes use of an unsupervised clustering algorithm that seeks to find the partition boundaries that "produce the greatest contrast" according to a given contrast function. The second method, referred to as mixed supervised/unsupervised, simply redefines the objective function to be maximized by dividing the previous contrast function by the entropy of a proposed partition. Since calculating the entropy for the candidate partition requires class label information, this

method can be thought of as supervised.

Dynamic programming methods have been applied to find interval boundaries for continuous features [113]. In such methods, each pass over the observed values of the data can identify a new partition on the continuous space based on the intervals already identified up to that point. This general framework allows for a wide variety of impurity functions to be used to measure the quality of candidate splitting points.

Vector Quantization [114] is also related to the notion of local discretization. This method attempts to partition an Nŋdimensional continuous space into a Voronoi Tessellation and then represent the set of points in each region by the region into which it falls. Alternatively, this method can be thought of as a complete instance space discretization as opposed to the feature space discretizations discussed here.

In theory, given a k-valued classification variable C, a continuous variable X could be partitioned into k sub-ranges by calculating e.g. Zeta [70] for each of the possible assignments of the k-1 cut points and selecting that combination of cut points that gives the largest value of Zeta. In general such a method will not be practicable because the number of combinations of cut points is extremely large.

[68] present two methods of cluster analysis: agglomerative bottom-up and divisive top-down. In agglomerative techniques, initially each case is a single cluster, and then they are fused together, forming larger and larger clusters. In divisive techniques, initially all cases are grouped in one cluster, then this cluster is gradually divided into smaller and smaller clusters. In both methods, during the first step of discretization, cluster formation, cases that exhibit the most similarity are fused into clusters. Once this process is completed, clusters are projected on all attributes to determine initial intervals on the domains of the numerical attributes. During the merging step adjacent intervals are merged together. The univariate case does not take into account any correlation between the explanatory attributes and fails to discover conjointly defined patterns.

While most discretization methods are employed as a preprocessing step to an induction algorithm, there are still other approaches, similar to the wrapper approach involved in the feature selection field. For example, the authors of [94] propose approaches that allow to refine the discretization of the continuous explanatory attributes by taking feedback from

an induction algorithm. Error-based methods, such as [124], evaluate candidate cut-points against an error function and explore a search space of boundary points to minimize the sum of false positive (FP) and false negative (FN) errors on the training set. In other words, given a fixed number of intervals, error-based discretization aims at finding the best discretization that minimizes the total number of errors (FP and FN) made by grouping together particular continuous values into an interval.

Elomaa and Rousu [111] showed that in order to find training set error (the optimal discretization in which the interval labeling differs least often from that of the data points) one only needs to examine a small number of all cut points, called alternation points. On the other hand, the authors prove that failing to check an alternation point may lead to a suboptimal discretization. Alternation points can be identified efficiently once the data has been ordered.

Elomaa and Rousu [112] presented techniques for speeding up the discovery of optimal with respect to an evaluation function multi-splits along numerical value ranges. The techniques are based on proving some cut points or prefixes of partitions as suboptimal and thus discarding them from the search space of the algorithms. Overall, the quadratic-time dynamic programming algorithm runs twice as fast on the average and in some cases up to 90 percent faster than the baseline algorithm.

3.16 Conclusion and Future Trends

Today, the discretization field is well studied in the supervised and unsupervised cases for an univariate process. However there is little work done in the multivariate case. It is virtually certain that better results can be obtained for a multivariate discretization if all attributes of the representation space are relevant for the learning problem.

Many basic problems still remain open with regard to discretization. The relationship between the nature of a learning task and the appropriate discretization strategies requires further investigation. One challenge is to identify what aspects of a learning task are relevant to the selection of discretization strategies. It would be useful to characterize tasks and discretization methods into abstract features that facilitate the selection.

Discretization for time-series data is another interesting topic. In time-series data, each instance is associated with a time stamp. The concept that underlies the data may drift over time, which implies that the appropriate discretization cut points may change. It will be time consuming if discretization has to be conducted from scratch each time the data changes. In this case, incremental discretization that only needs the old cut points and new data to form new cut points can be of great utility.

A third trend is discretization for stream data. A fundamental difference between time-series data and stream data is that for stream data, one only has access to data in the current time window, but not any previous data. The data may have large volume, change very fast and require fast response, for example, exchange data from the stock market and observation data from monitoring sensors. The key theme here is to boost discretizationŠs efficiency (without any significant accuracy loss) and to quickly incorporate discretizationŠs results into the learner.

As pointed out in earlier chapters, little or almost no work has been done before to aggregate discretization results in the form of bagging, boosting or perturb and combine methods.

Chapter 4

Discretization by Bootstrap

In this chapter we reintroduce the motivation behind our work and describe the various resampling based aggregation techniques for discretization that we have proposed and developed. We have proposed four techniques that are categorized into three types. All the approaches start with a random learning sample from a larger population and generate a fixed number of bootstrap samples with replacement and the solutions are build using these bootstrap samples.

4.1 Introduction

Our goal is to find a way to produce better discretization points. Previously, various studies have been done to estimate the discretization points from learning samples taken from the population. Because of inaccessibility of entire populations, we usually try to estimate statistical processes such as discretization from samples rather than the population. Significantly, in [31], a set of learning samples are used to approximate the best discretization points of the whole population. They argue that the learning sample is just an approximation of the whole population, so the optimal solution built on a single sample set is not necessarily the global one. Taking this point into consideration, in this paper we try to provide a better estimate toward the entire population. Our interpretation of the above problem leads us to use a resampling approach [79] to determine better distributions of the discretization points. In [96], the authors use various approaches including bootstrap to decrease discretization variance as preprocessing for decision trees. In this paper, we focus

on obtaining discretization points where each candidate point has a higher probability to be the "'better"' discretization point toward the whole population. By doing so, we attempt to improve on the predication accuracy of discretization and better estimation of the discretization points of the entire population, thus, treating the discretization problem in the statistical area with new results. We use *ordinary bootstrap* [9] as a method for resampling in our approach which tries to improve on the above mentioned problem. We argue that the recent increase in processing power of computers has allowed us to use extensive resampling analysis in order to find better estimates of the larger population.

In this paper we focus on supervised discretization, however, unsupervised discretization can also be applied in the same way. We present three different types of methodologies, all based on resampling by bootstrap, to obtain better quality discretization. Two of the methodologies consist of and estimation of the discretization point distribution over an attribute X_i by repeatedly resampling and performing discretization on each bootstrap sample using any of the discretization approaches and thus, creating a histogram density function of the obtained candidate discretization points. From this distribution we extract discretization points using two variant techniques explained in the following sections. The third methodology deals with obtaining discretization points while applying a non-hierarchical approach and building class frequency distributions for each class by repeated resampling the learning sample. Then the decision boundaries are identified as discretization points.

4.1.1 Case for Bootstrap

The bootstrap technique has been discussed in chapter two in detail. Because of inaccessibility of entire populations and the cost of extracting samples from them, we try to estimate statistical processes such as discretization thresholds from samples rather than the population. Parametric estimations are highly biased which has been explained in chapter 2. Among the nonparametric family the cost of extracting random samples from the population makes it not feasible thus, we adopt a bootstrap approach where the observed distribution of sample values is used as an estimate of the underlying probability distribution of the population. Then, the distribution for fixed sample sizes is obtained by repeatedly sampling by replacement from the initial sample, so that instead of individual partitions of the data

having the potential to occur more than once (like in random sampling), the individual values themselves may appear repeatedly in a single sample.

Even if we also samples to be extracted from the population, under this resampling algorithm the number of possible sample arrangements is much greater than for the randomization approach. Thus, any test statistic averaged across a series of say 500 samples under this algorithm will have a larger standard error since sub-samples of population can deviate more than under the randomization algorithm. Thus, our preference for bootstrap for building distributions for the following techniques is primary.

Here, we discuss two of the many commonly used bootstrap techniques.

4.1.1.1 Ordinary Bootstrap

Ordinary Bootstrap method [9] has the problem that the learning and test sets overlap. In this, a prediction rule is built on a bootstrap sample and tested on the original sample, averaging the misclassification rates across all bootstrap replications gives the ordinary bootstrap estimate. This method seriously underestimates the prediction error since a subset of data is used both in building and in assessing the prediction model.

4.1.1.2 Bootstrap Cross-Validation

The method is proposed by [62] to handle small sample problems. The procedure generates B bootstrap samples of size n from the observed sample and then calculates a leave-one-out cross-validation estimate on each bootstrap sample. Averaging the B cross-validation estimates gives the bootstrap cross-validation estimate for the prediction error. Since an original observation can appear more than once in a bootstrap sample, a leave-one-out learning set may overlap with the left out item when the cross-validation procedure is applied on a bootstrap sample. Consequently, the bootstrap cross-validation method tends to underestimate the true prediction error.

There are other techniques such as the .632 bootstrap by Efron and Tibshirani [9], Leave-out-one bootstrap by Efron [8] and the out of bag estimation [77]. Most of the bootstrap methods like the ones discussed above have an increased danger of overfitting to noise in the data, a problem which may be addressed by combining bootstrapping methods with

cross-validation.

In all our solutions we use a k-fold cross-validated bootstrap like technique. We generate B bootstrap samples of size n from k-1 folds and then test each bootstrap sample on the remaining fold(s).

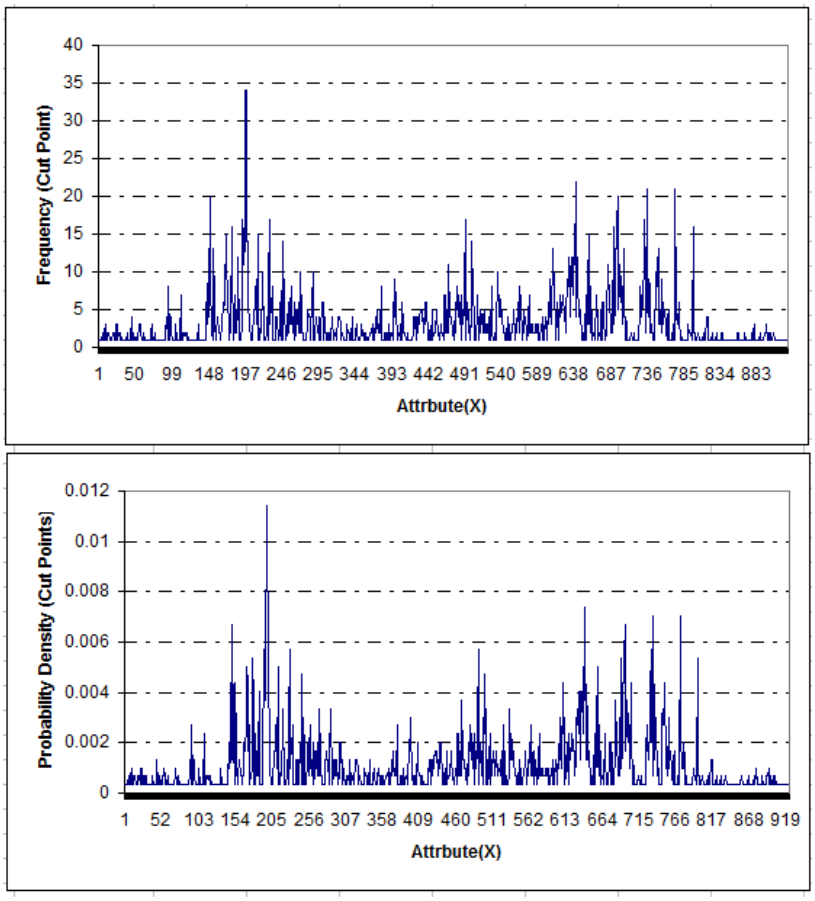

Figure 4.1: Histogram Density Function obtained from repeated discretizations.

4.1.2 Assumptions

We build our discretization keeping in mind the following assumptions.

1. In each interval, the distribution of the class values is defined by the frequencies of the class values in this interval.

2. The attributes are conditionally independent of each other given the class.

3. Thus, we discretize one attribute at a time with the class (in two dimensions).

In the following sections we describe the three methodologies that we have used for discretization.

4.2 Discretization Point Distribution based Approach

Earlier, we argued that the learning sample is just an approximation of the whole population, so the optimal discretization solution built on a single sample set is not necessarily the global one. Thus, in this section we present a technique which is based on finding a better discretization estimate towards the entire population in terms of discretization quality (as discussed before), using a sample selected randomly from that population and then further resampling it B times by using a k-fold cross-validated bootstrap like technique (discussed in section 4.1.1) to achieve this objective.

From the B bootstrap samples we generate discretization points by applying a discretization procedure $disc(X(\Omega), d_{type})$ on each bootstrap sample Ω_{BS}. These B repeated discretizations generate $\psi = e_1, e_2, ..., e_J$ discretization or cut points. Then, from these aggregated discretization points we build a probability distribution of these points as shown in figure 4.1. The fiqure 4.1(top) shows the probability of occurrence for each point among J cut points obtained from b multi-interval discretizations and figure 4.1(bottom) shows the histogram density function $H(\psi)$ obtained from this distribution. From this aggregated discretization point probability distribution we choose the most probable p discretization points, which we argue, shall improve the quality of discretization from the ones obtained from a single random sample. Thus, our technique relies on the hypothesis that the distribution build by bootstrap is very close to the true distribution of the discretization points.

The probability distribution of discretization points is a histogram density function $H(\psi)$ and is built as:

$$H(\psi) = (\xi_j, f_j)/\xi_j \in \psi : f_j = \frac{card(e_k \in \psi / e_k = \xi_j)}{card(\psi)}$$

Where $j = 1, ..., J$, $\psi = e_1, e_2, ..., e_g$ are all the discretization points obtained from B discretizations and $i = 1, ..., g$.

The best discretization consists of finding the p most probable discretization points from this distribution. Thus, the problem remains of finding the best p or in other words the best 'number of intervals', which is achieved by the following discretization techniques. These two solutions are explained in below in detail:

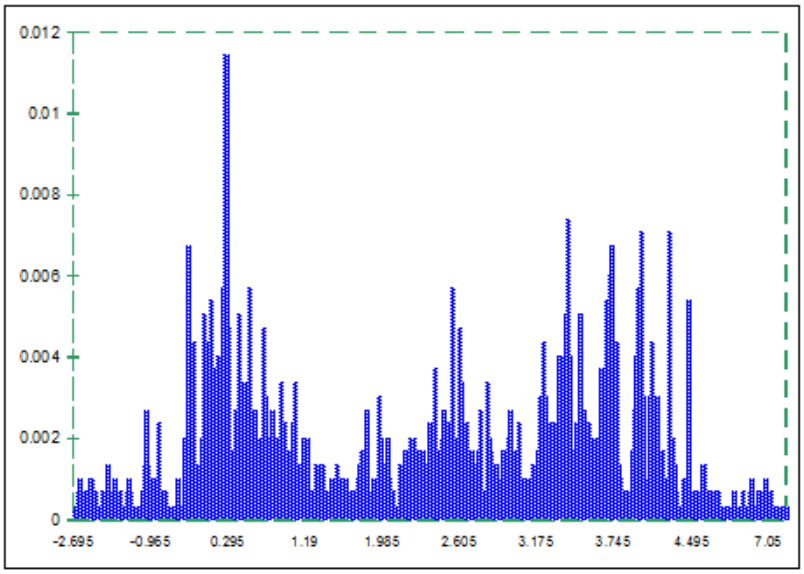

Figure 4.2: Histogram the obtained discretization or threshold points exhibiting high threshold variance.

4.2.1 Variance Study

In this sub-section we introduce two new types of variance in addition to the typical bias/variance decomposition of a learning algorithm explained in chapter 2. We denote threshold variance indicating the variance of the discretization threshold point or cut point with multiple samples. This threshold variance is easy to demonstrate if we have only one threshold point or cut point. In the case of multiple cut points we shall fix the number of intervals and calculate the deviations from the mean threshold for each interval. Another

type of variance we are interested in studying is the variance of the accuracy with each changing sample. This is to see how much the accuracy rate of each obtained discretization for each sample deviates from the average accuracy.

In order to demonstrate the problem of high discretization threshold variance, we generate 1000 random samples of size 300 from the attribute number 7 of the waveform dataset obtained from the UCI data repository [15]. This attribute has values ranging from -2.65 to 8.76. We applied a top-down entropy based discretization, limiting it to 3 discretization points or 4 intervals per sample. Fig 4.2 shows the probability distribution of the obtained discretization points showing a generally flat distribution illustrating the high variance of these threshold points with each random sample. The average variance of the 3 threshold discretization points was calculated as $\sigma_t = 1.087$, which is quite high. Here, we also calculated the variance of the prediction accuracy achieved by the discretization performed on each sample. This was estimated as $\sigma_p = 8.58$, again a high deviation from the mean accuracy.

4.2.2 Definitions

4.2.2.1 Smoothing

We use a moving average filter as a smoothing technique which is the unweighted mean of the previous n data points.

4.2.3 Resampling based Discretization Point Distribution based Technique (RDD)

This technique is carried out in two phases. In the first phase we estimate a discretization point distribution over an attribute X_i by repeated resampling n times and performing discretization on each bootstrap sample Ω_{BS} using an entropy based MDLPC method and thus, creating a histogram density function of the resulting candidate points as shown in fig 1a. Note that we can use any discretization method in place of MDLPC but in our experiments we happen to use this technique. During the discretization performed on each bootstrap sample, we record the number of intervals for each sample and build the histogram density function for the number of intervals obtained for n bootstraps. Then from this

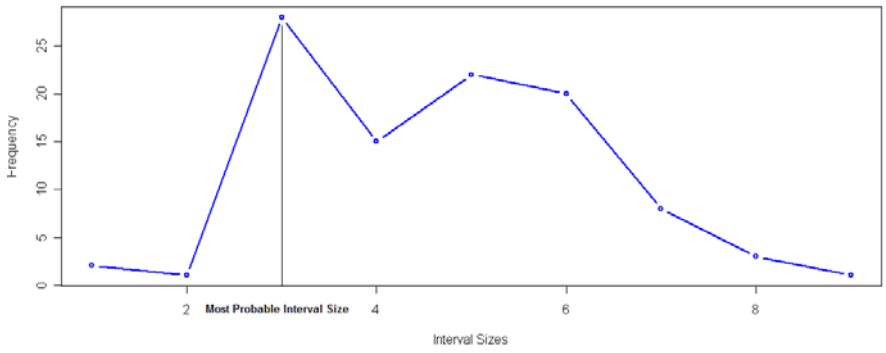

Figure 4.3: Number of intervals (interval frequencies) distributed in an example data.

interval distribution we select the most probable interval obtained denoted as I_{best}. This is illustrated in figure 4.3.

In the second phase we select the I_{best} most probable discretization points i.e. those I_{best} number of points which have the highest frequencies. These points shall be our discretization points. This algorithm is listed below:

4.2.3.1 Algorithm 1.1 - RDD

1. Create a learning sample $X(\Omega_{LS})$.

2. From $X(\Omega_{LS})$, generate B bootstrap samples $X(\Omega_{BS_1}), ..., X(\Omega_{BS_B})$ of size B_{size}.

3. For each $i = [1, ..., B]$; $V_i = Disc(X(\Omega)_{BS_i})$, where ψ_i is a vector containing the discretization points obtained from a discretization on a sample $X(\Omega_{BS})$ and $I_i = card(\psi_i)$, which is the number of discretization points generated by the discretization.

4. Create a histogram of the frequencies of discretization points and then a histogram density function:

$$H(\psi) = (\xi_j, f_j)/\xi_j \in \psi : f_j = \frac{card(e_k \in \psi / e_k = \xi_j)}{card(\psi)}$$

Where $j = 1, ..., J$, $\psi = e_1, e_2, ..., e_g$ are all the discretization points obtained from B discretizations and $i = 1, ..., g$.

5. Also create a histogram of the frequencies of the number of intervals for each discretization and then a histogram density function:

$$H(\kappa) = (I_j, f_j) : f_j = \frac{I_j}{card(\kappa)}$$

Where $\kappa = I_1, I_2, ..., I_J$ are the number of intervals and $j = 1, ..., J$.

6. Set $I_{max} = max(H(\kappa))$ which is the maximum interval frequency obtained for different discretizations i.e the highest probable number of discretization points obtained among B discretizations.

7. Select the top I_{max} highest probable discretization points from the discretization point distribution $H(\psi)$, such that: $j = 1, ..., I_{max}$:

$$f_j = \begin{Bmatrix} max(f_j)), & if\ j{=}1) \\ max(f_j){:}f_j{<}f_{j-1}), & Otherwise \end{Bmatrix}$$

Where, $d_j = (\xi_j, f_j)$ i.e. the values ξ_j are the selected discretization points with the interval size $I_{max} + 1$.

4.2.4 Resampling based Smoothed Discretization Point Distribution based Technique (RSDD)

This technique is carried out in three phases. The first phase is the same as in the RDD technique explained above. In the second phase we smooth over the discretization point distribution function by applying a moving average filter with a window size ws. The resulting smoothed curve is shown in fig 1b. We can see distinct regions (or peaks) where the probability of the candidate points to be the exact discretization point is higher. As shown in fig 1b, the straight line that runs parallel with the x-axis is called the threshold parameter T, which is set as the median of the obtained frequency values. By taking into account this threshold T we define peaks P_{reg} as the regions which lie above this threshold line. The reason for defining such a threshold is to help to elaborate only the regions of higher probabilities (most frequent). The ws is determined as follows: We start by setting $ws = 3$ and we calculate the number of peaks P_{reg} obtained. We continue to increment the

ws until the the number of peaks P_{reg} obtained approach the most probable interval number I_{best} as above.

In the third phase, we extract candidate discretization points from this smoothed function. For this phase we have two variations; (1) applying a voting procedure to the peaks(fig 1b) or (2) averaging the peaks.

1. From each peak P_{reg} we select or vote the most probable or frequent point as shown in fig 4.4b.

2. For each peak P_{reg} we take the average of all the points that lie in the peak region and select the point on the x-axis that corresponds to that average.

4.2.4.1 Algorithm 1.2 - RSDD

1. Create a learning sample $X(\Omega_{LS})$.

2. From $X(\Omega_{LS})$, generate B bootstrap samples $X(\Omega_{BS_1}), ..., X(\Omega_{BS_B})$ of size B_{size}.

3. For each $i = [1, ..., B]$; $V_i = Disc(X(\Omega)_{BS_i})$, where ψ_i is a vector containing the discretization points obtained from a discretization on a sample $X(\Omega_{BS})$ and $I_i = card(\psi_i)$, which is the number of discretization points generated by the discretization.

4. Create a histogram of the frequencies of discretization points and then a histogram density function:

$$H(\psi) = (\xi_j, f_j)/\xi_j \in \psi : f_j = \frac{card(e_k \in \psi / e_k = \xi_j)}{card(\psi)}$$

   Where $j = 1, ..., J$, $\psi = e_1, e_2, ..., e_g$ are all the discretization points obtained from B discretizations and $i = 1, ..., g$.

5. Smoothen the above distribution by applying a moving average filter with a window size of W_s such that for $i = 1, ..., J$,

$$H(\psi) = \left\{ \begin{matrix} f_i), & if\ i<W_s \\ \frac{1}{ws}\sum_{j=(i-W_s)}^{i} f_j), & otherwise \end{matrix} \right\}$$

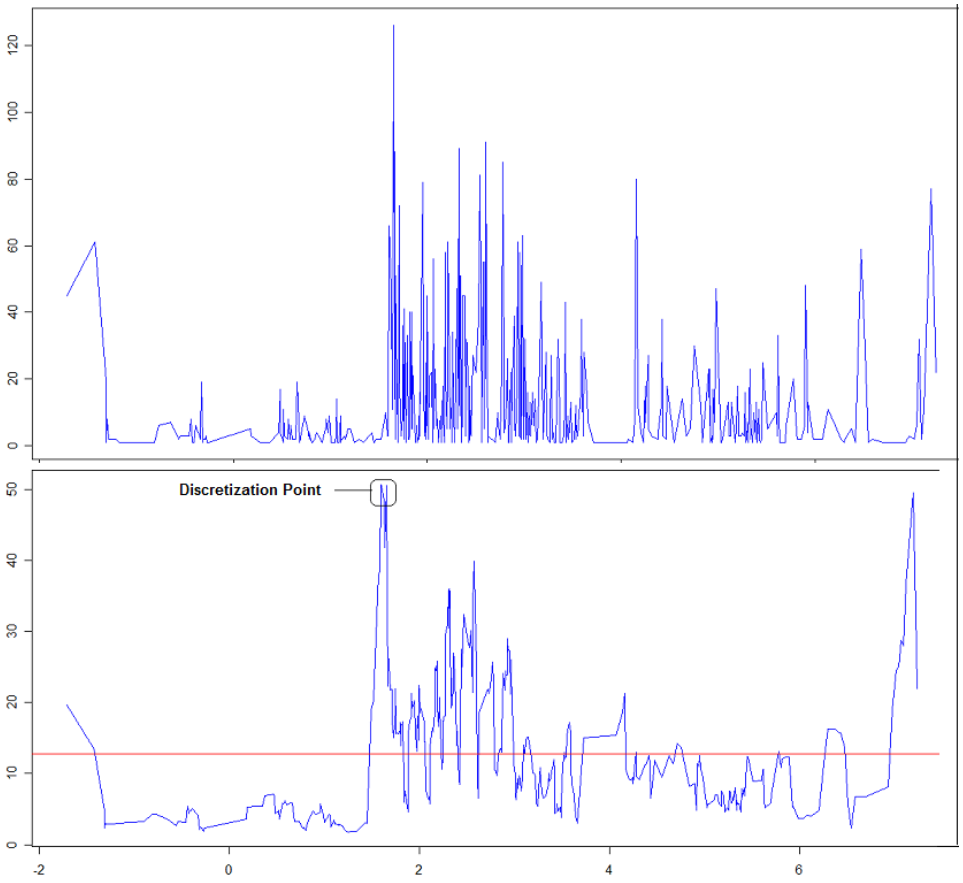

Figure 4.4: **a)** Discretization point frequency distribution of variable 1 of the waveform data where the x-axis represents the attribute's continuous values and the y-axis represents their frequencies. **b)**. The distribution of fig 4.4a after smoothing with a $ws = 7$ and the resulting peaks.

6. Set a threshold $\forall jin1, ..., J : t = mean(f_j))$.

7. Identify points that lie on or close to the threshold t, such that,

   $\forall j = 1, ..., J$, the threshold points are those points among the discretization points $d_j$such that $tpoints_j = \begin{Bmatrix} f_j, & if\ |t-f_j|-|t-f_{j+1}| \leq 0 \\ f_{j+1}, & if\ |t-f_{j+1}|-|t-f_j| \end{Bmatrix}$, iff $f_j < t\ and\ f_{j+1} \geq t$ or $f_j \geq t\ and\ f_{j+1} < t$.

8. Next we identify the points contained in the peak regions as described previously. We define each peak region by a vector $Peak_j$ such that,

$\forall i = 1, ..., t_s$ and $l = 1, ..., J$ set $Peak_j = f_j > t : tpoints_i \leq f_l \leq f_{l+1}$, where t_s is the number of threshold points.

9. For each $Peak_j$, each discretization point is the most probable point contained in the peak region such that, $d_j = (\xi_j, max(Peak_j))$.

4.3 Top-Down Discretization by Bagging

In this section we adopt a slightly different approach towards the utilization of a bootstrap based aggregation technique. This technique is a variant of the popular 'Bagging' [76] method explained in detail previously in chapter 3. Our motivation for this approach stems for the work of the authors in [96], where they use various approaches including voting and bagging like techniques to decrease discretization variance as preprocessing for decision trees. In summary they obtain a single discretization threshold for each bootstrap sample and build a threshold frequency distribution by repeated resampling (by bootstrap). Then they vote for the most frequent threshold point as the cut point for the decision tree.

However, they only cater for single threshold or discretization point having only two intervals. Whereas, we try to extend this concept in order to improve the discretization quality of top-down discretization techniques with multiple intervals. Although, Bagging has been successfully used before [96] in improving the prediction accuracy and variance of a classifier such as decision trees, but to our knowledge it has not been used to improve multi-interval discretization.

Consider a top-down discretization built on a sample. Instead of selecting the threshold point d^t that improves a certain criteria (i.e entropy in the case of MDLPC), we bootstrap multiple times and aggregate by voting for the most frequent threshold point d^t_{best} as formalized in the following equation. In this way, we split the sample into two and continue until a stopping criteria is reached.

$$d^t_{best} = arg\ max\ \sum_{i=1}^{B} (Split(\Omega_{BS_i}) = d^t_i)$$

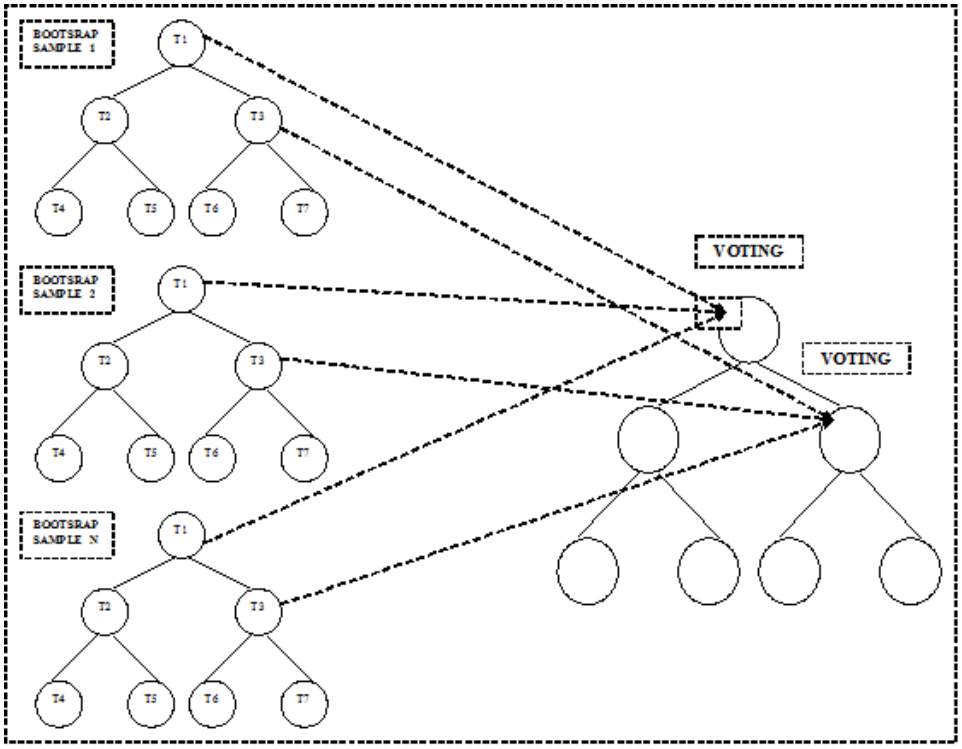

Figure 4.5: The left hand side represents n top-down binary trees generated from n bootstrap samples with q nodes representing the threshold points t. The right hand side represents the final tree after voting for the best split at each node.

4.3.1 Threshold Bagging based Top-Down Discretization Technique (TBTD)

Our approach is based on finding a better discretization estimate toward the entire population in terms of discretization quality (as discussed above), using a sample selected randomly from that population and then resampling it. We aggregate and vote for the best split threshold point d^t_{best} at each node N and use a bootstrap based Bagging technique to achieve this objective. The solution is explained below:

This technique is carried out in three phases. In the first phase, we obtain B bootstrap samples from the training or learning set Ω_{LS} of the same size and generate B threshold points on each bootstrap sample as shown in figure 4.5. In phase two, we try to aggregate and select the best threshold point d^t_{best} by using a voting method. This is done by building a histogram distribution of the B threshold points for each node N as shown in figure 4.6 and

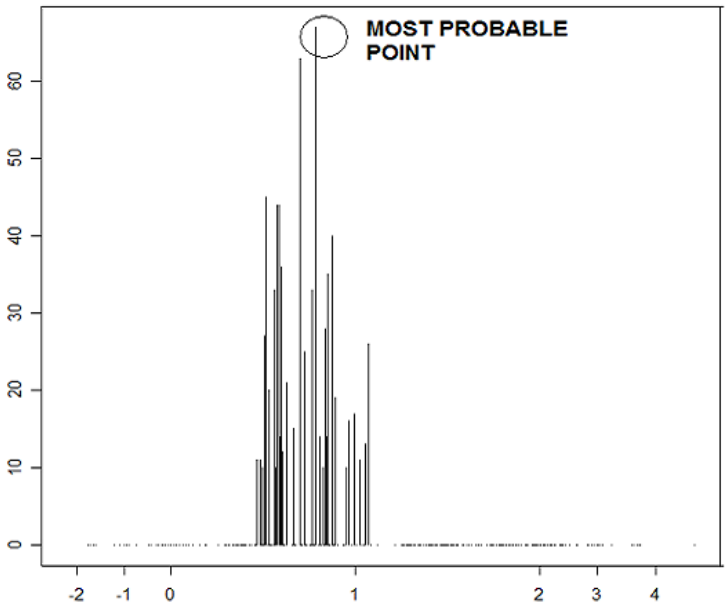

Figure 4.6: Distribution of threshold points t at a node n obtained from n bootstrap results.

selecting the most probable or frequent threshold point as the final threshold d^t_{best}. Next, in the third phase we split the original learning sample Ω_{LS} at the threshold point d^t_{best} and obtain two new resulting sample. This recursive partitioning process comprising of the above three phases, is repeated for each new node representing a subsample of the original sample Ω_{LS}. We can deploy any stopping rule used by any existing top-down method (criteria) being used e.g. we use the stopping criteria of the MDL principle as explained in chapter 3.

4.3.1.1 Algorithm 1.3 - TBTD

1. Create a learning sample $X(\Omega_{LS})$.
2. From $X(\Omega_{LS})$, generate B bootstrap samples $X(\Omega_{BS_1}), ..., X(\Omega_{BS_B})$ of size B_{size}.
3. For each $i = [1, ..., B]$; $d^t_i = Split(X(\Omega)_{BS_i})$, where d^t_i is the best threshold split point obtained from maximizing a certain criteria on a sample $X(\Omega_{BS})$.
4. Create a histogram of these B threshold points i.e. $d^t_1, d^t_2, ..., d^t_B$.

5. Retain the most frequent threshold split point from the histogram as $d^t_{best} = max(d^t_i)$ where $i = [1, ..., B]$.

6. Cut the learning sample $X(\Omega_{LS})$ at this point and we get two new sub-populations.

Repeat from step 2, the same procedure for each sub-sample, which begins by bootstrapping B times until a stopping criteria is met.

4.4 Class Distributions by Resampling

In this section we make an attempt to exploit the notion of decision boundaries (definition 4.41) generated from class distributions. Here, we try to simplify and find an alternative process of generating discretization points using a class dependent or supervised methodology. The concept of decision boundaries is explained in section 4.4.1 and illustrated in figure 4.5. In short they are the intersection of any two class distribution functions of an attribute X. These decision boundaries represent the change in the domination or probability of one class function over the other. This concept is explained in the following example.

Assume a discretization method to create intervals $I_i; (i = 1, ..., 5)$ as in Figure 4.7. I_2 and I_4 each contain a decision boundary while the remaining intervals do not. For any two values in I_2 (or I_4) but on different sides of a decision boundary, the optimal naive-Bayes learning under zero-one loss should select a different class for each value. But under discretization, all the values in the same interval cannot be differentiated and we will have the same class probability estimate for all of them. Consequently, a classifier such as naive-Bayes with discretization will assign the same class to all of them, and thus values at one of the two sides of the decision boundary will be misclassified with respect to the optimal classification under zero-one loss. The larger the interval frequency, the more likely that the value range of the interval is larger, thus the more likely that the interval contains a decision boundary. The larger the interval containing a decision boundary, the more instances to be misclassified, thus the greater the expected bias of the generated classifiers. In other words, larger interval frequency tends to incur higher discretization bias. Consequently, discretization bias can be reduced by identifying the decision boundaries and setting the interval boundaries close to them. However, identifying the correct decision boundaries

depends on finding the true form of $p(C|X_i)$ for each quantitative attribute X_i. Ironically, if we have already found $p(C|X_i)$, we can resolve the classification task directly; thus there is no need to consider discretization any more.

Without knowing $p(C|X_i)$, a solution is to increase the interval number so as to decrease the interval frequency. An extreme approach is to set each (different) value as an interval. Although this most likely guarantees that no interval contains a decision boundary, it usually results in very few instances per interval. As a result, the estimation of $p(C = c|X_i^* = x_i^*)$ for each c might be so unreliable that we cannot identify the truly most probable class even if there is no decision boundaries in the interval. The larger the interval number for probability estimation, the greater the expected variance of the generated classifiers, since even a small change to the training data might substantially change the probability estimation. In other words, larger interval number tends to incur higher discretization variance.

Thus, in order to estimate $p(C|X_i)$, we take the learning sample and generate then aggregate B bootstrap samples. From this aggregate resampled data we build class distributions $p(C = c|X_i)$ for each class c by considering the percentages of class at each value x_i of X. Thus, we plot a histogram density function $f(X_i|C = c)$ of each class and overlay them as shown in figure 4.7. Then we smooth out these distribution curves and identify decision boundaries from this estimation which, are kept as discretization points. Thus, the region between each decision boundary is assigned the majority or dominant class.

4.4.1 Definition - Decision Boundary

Hsu et al [49] addressed this factor in the context of estimating the probability density function $f(X_i|C = c)$ of a quantitative attribute X_i given each class c. They defined decision boundaries of X_i as intersection points of the curves of $f(X_i|C)$, where ties occurred among the largest conditional densities. In our case these probability density functions are built from resampling the sample data and building (then smoothing) a histogram density function based on class frequencies.

However, this definition is further refined by Yang et al [140] where they first define a most probable class. When classifying an instance x, a most probable class c_p given x is the class that satisfies $\forall c \in C;\ P(c|x) \leq P(c_p|x)$. Note that there may be multiple most

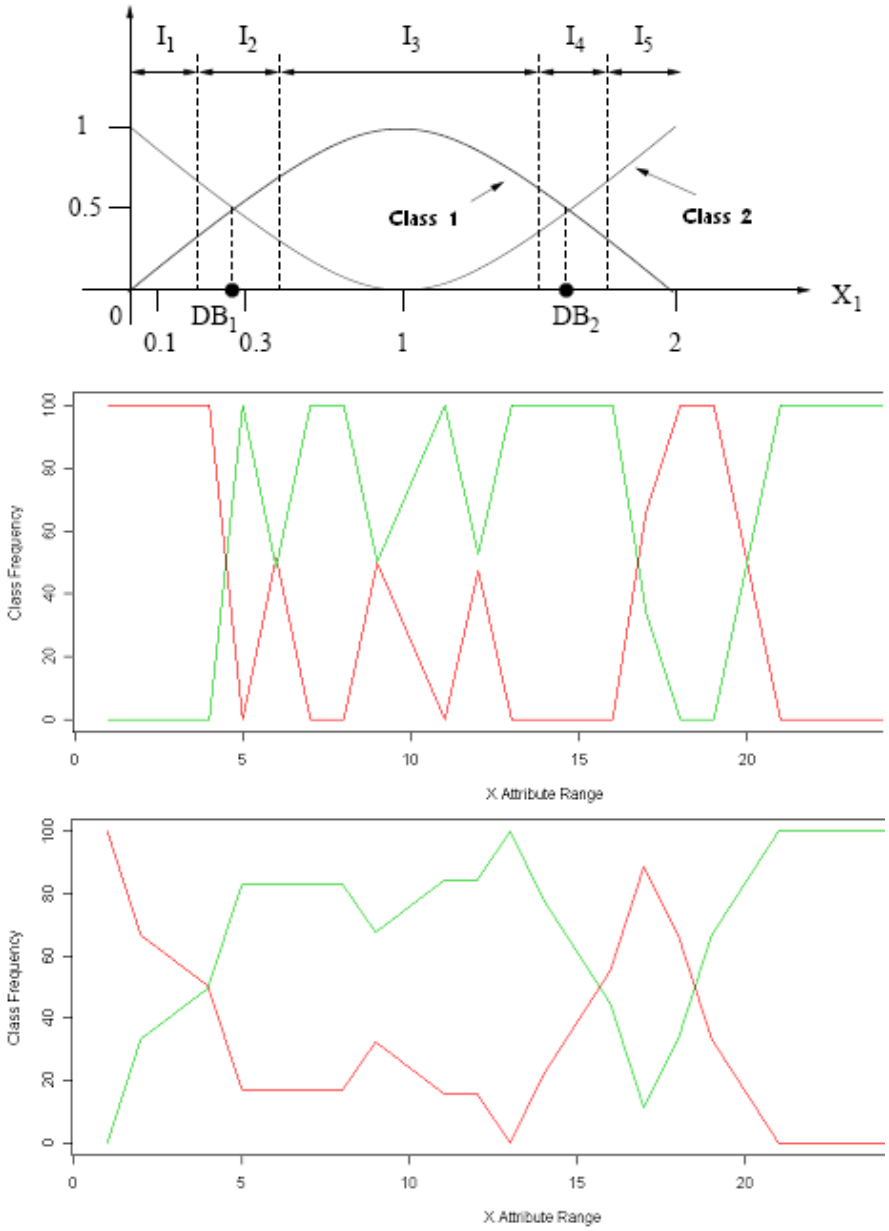

Figure 4.7: Decision boundaries for class 1 and 2.

probable classes for a single x if the probabilities of those classes are equally largest. In consequence, they define a set of most probable classes whose elements are all the most probable classes for a given instance x. We use $C_{mp}(x)$ to represent the set of most probable classes for x.

A decision boundary of a quantitative attribute X_i given an instance x is an interval (y, z) of X_i (that may be of zero width) such that:

$$\forall(w \in [y; z), u \in (y; z]), \neg(w = y \wedge u = z) \Rightarrow C_{mp}(x\ X_i = w) \cap C_{mp}(x\ X_i = u) \neq 0$$

and

$$C_{mp}(x\ X_i = y) \cap C_{mp}(x\ X_i = z) = 0$$

The consequent decision boundaries are shown in fig 4.7, labelled as DB1 and DB2 respectively. The most probable class for a value x_1 changes each time its location crosses a decision boundary.

4.4.2 Resampled Decision Boundary based Technique (RDB)

Our proposed discretization technique known as RDD comprises of the following steps:

- We take a random data sample Ω_{rs} from the entire population.
- From this data sample Ω_{rs}, we generate a large data sample Ω_{bs} by repeated resampling of Ω_{rs}, $n = 1000$ times by using ordinary bootstrap.
- Next, we create an ensemble histogram density function of the attribute X_i (to be discretized). This is achieved by merging the plots of all the class frequency histograms (percentage frequencies) as shown in fig 4.7(middle).
- Then we apply a smoothing procedure with $ws = 7$ to each class frequency curve as illustrated in fig 4.7(bottom).
- Finally, our discretization points are the decision boundaries which are the intersection of the curves of any two or more classes. This is illustrated in fig 4.7(bottom).

By building a collective histogram frequency distribution of all the classes, we try to obtain a better estimate of the class distribution of data relative to the entire population. Thus, this can be termed as a prior distribution or a probabilty density function and its application can extend also in *naive bayesian classifiers* [140].

4.4.2.1 Algorithm 1.4 - RDB

1. Create a learning sample $X(\Omega_{LS})$.

2. From $X(\Omega_{LS})$, generate B bootstrap samples $X(\Omega_{BS_1}), ..., X(\Omega_{BS_b})$ of size n.

3. Build a distribution $f(X|C)$ for each class $c_1, ..., c_m$ on each bootstrap sample and take the average of $f(X|C)$ at each point on $X = x_1, ..., x_a$ such that $\forall X$,

$$f(X|c_i) = \frac{1}{b}\sum_{j=1}^{b} f_j(X|c_i), \quad \text{where } f_j(X|C = c_i) = \frac{card(C(w)=c_i)}{card(X(w))} \text{ for } i = 1, ..., m.$$

4. Smoothen the above distributions $f(X|c_i)$ by applying a moving average filter with a window size of ws such that for $i = 1, ..., a$ and $\forall C$,

$$f(x_i|c) = \left\{ \begin{matrix} f(x_i|C), & if\ i<ws \\ \frac{1}{ws}\sum_{j=(i-ws)}^{i} f(x_j|C), & otherwise \end{matrix} \right\}$$

5. Overlay all the class probability distributions $f(X|c_i), ..., f(X|c_m)$.

6. The intersection points of any two or more class distributions are the discretization points.

4.5 Conclusions for this Chapter

In this this chapter we presented 3 different methods in which we use resampling based bootstrap techniques in order to try and obtain better discretization points. Among our methods the first technique based on building a resampled (B times) discretization point distribution has further two variations namely RDD and RSD. These two methods vary in the manner they extract selected discretization points from the built distribution. The former selects the most probable n points. This number n is calculated by considering the most frequent 'number of intervals' obtained by the earlier repeated resampling. Whereas, the later smooths the distribution by using a moving average filter and then identifies 'peak regions' on most probable regions in the distribution. The most frequent points in each of these 'peak regions' are selected as discretization points.

The second method TBTD, works on top-down discretizations and applies a Bagging technique at each node (subsample) and selects the best point by a voting procedure. The original sample is split at this point and the sample procedure is recursively applied to the next two subsamples until a stopping criteria is met.

The third method builds probability density functions $f(X|C)$ of class distributions on an attribute X by resampling the learning sample. It further smooths the class distributions and identifies the decision boundaries as the final discretization points. This method is more suitable for naive-bayes classifiers because of its tendency to generate a larger number of intervals.

Chapter 5

Experimentation and Results

In this chapter we perform detailed experiments on various discretization techniques and compare their performances with our aggregative discretizations. In doing so we provide analysis and results on each experiment done and try to deduce reasonable conclusions.

5.1 Evaluated Discretization Techniques in Brief

In addition to our aggregated discretization techniques, we have evaluated six other discretization methods which have been described in sufficient detail in chapter 3. Out of these three methods use a top-down technique such as (MDLPC, Fusbin, BalancedGain) and two are bottomup (Chimerge, MODL), while one is based on an optimal algorithm (Fisher) described in section 3.7. A summary of these methods is given below:

The MDLPC method is a greedy top-down split method, whose evaluation criterion is based on the Minimum Description Length Principle [121]. At each step of the algorithm, the MDLPC evaluates two hypotheses (to cut or not to cut the interval) and chooses the hypothesis whose total encoding cost (model plus exceptions) is the lowest. The Balanced-Gain method exploits a criterion similar to the GainRatio criterion [67]: it divides the entropy-based Information-Gain criterion by the log of the arity of the partition in order to penalize excessive multisplits. This method can be embedded into a dynamic-programming based algorithm, as studied in [110]. FUSBIN adopts a quadratic entropy criteria based on the uncertainty principle and the idea of penalizing according to the size of the sample.

Dataset	Continuous Attributes	Size	Class Values
Adult	7	48842	2
Australian	6	690	2
Breast	10	699	2
Crx	6	690	2
Heart	10	270	2
Hepatitis	6	155	2
Hypothyroid	7	3163	2
Iris	4	150	3
Pima	8	768	2
Waveform	21	5000	3

Figure 5.1: Data sets and their summary.

The ChiMerge method is a greedy bottom-up merge method that locally exploits the chi-square criterion to decide whether two adjacent intervals are similar enough to be merged. MODL [81] algorithm is based on a bayesian approach. It defines a criterion which is minimal for the bayes optimal discretization. Fisher's algorithm is a dynamic programming algorithm that tries to find the optimal partition given a criterion, introduced by W.Fisher [122]. The criterion that is used in this paper for Fisher's algorithm is based on Fusinter's uncertainty principle, that exploits an uncertainty measure sensitive to the sample size. Its criterion employs a quadratic entropy term to evaluate the information in the intervals and is regularized by a second term in inverse proportion of the interval frequencies.

5.2 Data Sets

All the experiments have been done using the ten data sets from the UCI repository [15] laid out in fig 5.1. In all, these ten datasets have 85 continuous attributes denoted as $(X_1(\omega), ..., X_85(\omega))$ and a label $C(\omega)$. These datasets comprise of 2 or 3 classes and contain small and large sets, with Iris data having 150 examples and the largest Waveform data containing 5000 examples. Note that the waveform dataset contains significant noise as compared to the others.

5.3 Results - Analysis and Comparisons

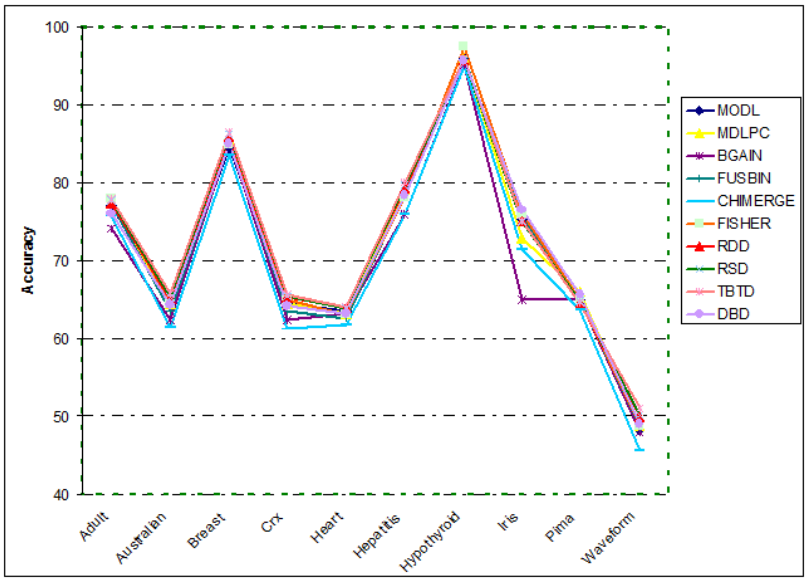

Figure 5.2: Rough Comparison of the Accuracy of all the methods for each dataset.

Discretization methods have been evaluated both using decision trees [119, 59] and naïve Bayes methods [49]. In their experiments, Kohavi et al [99] report that entropy-based discretization methods perform better that error-based methods. Elomaa et al [110] show that using optimal multi-splitting discretization algorithms does not bring a clear accuracy advantage over binary splitting in the case of decision trees. However, Fayyad et al [118] show that multi splitting can be more effective in terms of accuracy and other aspects than the binary counterparts. In the case of naïve Bayes classifiers, Dougherty et al [59] demonstrate that using any discretization algorithm outperforms the naïve Bayes algorithm with the normality assumption for continuous attributes.

Although these evaluations bring many insights on the impact of discretization methods on classifiers, the contribution of the discretization is not always clear [81]. The reason is that discretization is a preprocessing step for classifiers such as decision trees but it can be done once at the root of the tree or repeated at each node of the tree. It can use a binary-

splitting strategy or a multi-splitting strategy and so, the performances of such classifiers result from complex interactions between various types of these and other strategies.

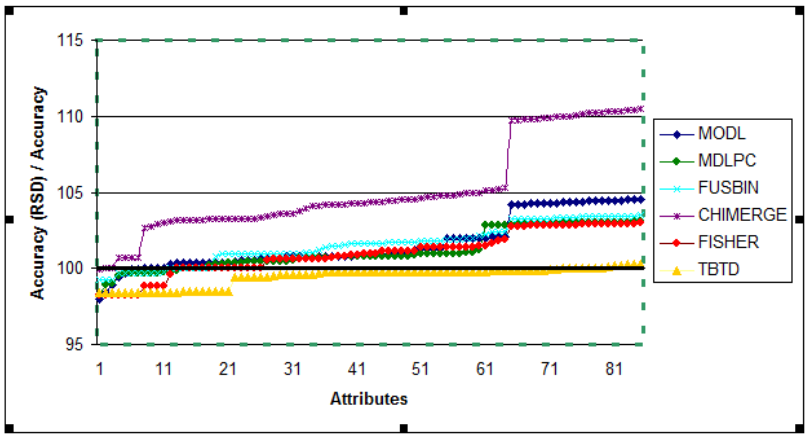

Figure 5.3: Repartition function of RSD vs all the other methods.

In order to evaluate the intrinsic performance of the discretization methods and eliminate the bias of the choice of a specific induction algorithm, Zighed et al [31] consider each discretization method as an elementary inductive method that predicts the local majority class in each learned interval. They apply this approach on the waveform dataset to compare several discretization methods on the accuracy criterion. This has been discussed in detail in chapter 3. This same approach is also used by Boullé [81] where many detailed experiments have been performed. We extend this protocol and evaluate the elementary discretization inductive methods using all the continuous attributes contained in several datasets. This allows us to perform hundreds of experiments instead of at most tens of experiments. The discretizations are evaluated for three criteria: accuracy, robustness (test accuracy/train accuracy) and number of intervals.

Most of the time we used a 3-fold cross validation to measure the geometric mean of the accuracy rate, bias/variance decomposition, number of intervals and robustness (defined in chapter) from discretization of all the 85 variables using all the 6 methods discussed. However, in some experiments we divided the data into two sets i.e. a smaller learning set and a larger test set. Then we calculated RDD, RSD, TBDT and DBD as described in chapter

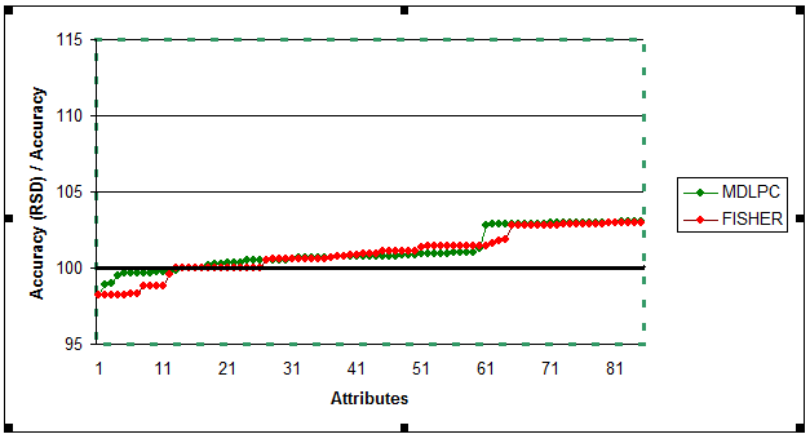

Figure 5.4: Repartition function focusing on RSD vs MDLPC and Fisher.

4. The resulting measures and their results are explained in the following subsections. In our experimentations we obtained RDD, RSD and TBDT techniques using the MDLPC discretization obtained from $n = 100$ bootstrap samples and try to build a near optimal solution. Then, we compare this solution to MDLPC original and other five discretization methods defined above. For the DBD method we adopted the use of 500 bootstrap samples. All the above discretization methods have been implemented using the software R (www.r-project.org/).

The MODL, MDLPC and BalancedGain methods have an automatic stopping rule and do not require any parameter setting. For the Fusinter criterion, we use the regularization parameters recommended in [29]. For the ChiMerge method, the significance level is set to 0.95 for the chi-square test threshold. For Fisher's optimal algorithm we choose the criteria used in FusInter [31]. In order to find the add-value of our resampling based discretization techniques and compare them with the above mentioned top-down and bottom-up strategies, we measure the quality of discretization and comparing the strategies considering ten different datasets shown in figure 5.1.

t*	RSD	TBDT	RDD	DBD	MDLPC	MODL	FUSBIN	CHIMERGE	BGAIN	FISHER
RSD	X	-2.57	3.1	9.17	9.22	10.38	11.1	14.8	13.5	6.87
TBDT		X	5.67	11.74	11.79	12.95	13.67	17.37	16.07	9.44
RDD			X	6.07	6.12	7.28	8	11.7	10.4	3.77
DBD				X	0.05	1.21	1.93	5.63	4.33	-2.3
MDLPC					X	1.16	1.88	5.58	4.28	-2.35
MODL						X	0.72	4.42	3.12	-3.51
FUSBIN							X	3.7	2.4	-4.23
CHIMERGE								X	-1.3	-7.93
BGAIN									X	-6.63
FISHER										X

Figure 5.5: Comparison of the critical area between all the methods.

5.3.1 Predication Accuracy

We consider the prediction accuracy of an obtained discretization over an attribute as the most important criteria. Figure 5.2 illustrates a rough estimate of the prediction rates of all the discretization methods plotted against all the data sets. The resulting curves are quite close to each other and hence, are not that clear. Thus, we choose other representations as well. However, upon careful observation we can see that the curve of TDBT, RDD and RSD most of the time, lie above all the other methods including MDLPC from which our techniques are originally built (by resampling and obtaining selected discretization points) as explained earlier. On the contrary, the curves of Chimerge and Bgain lie obviously often below the others.

In order to analyze the relative differences of accuracy for the 85 attributes in more details, we collect all the geometric mean ratios per attribute in ascending order. Figure 5.3 shows the repartition function of the relative differences of accuracy between the RSD method and the other discretization methods. Each point in this repartition function is the summary of 10 discretization experiments performed on the same attribute. Figure 5.4 illustrates the comparison between RSD, MDLPC and Fisher's optimal algorithm using FUSINTER.

Such repartition functions have been used in [81] and represent a convenient tool for the fine grain analysis of the differences between methods, in complement with the multi-criteria analysis carried out on the coarse dataset geometric means. A flat curve reflects two methods that do not differentiate on any of the experiments. A symmetric curve correspond to methods that globally perform equally well, but with differences among the experiences.

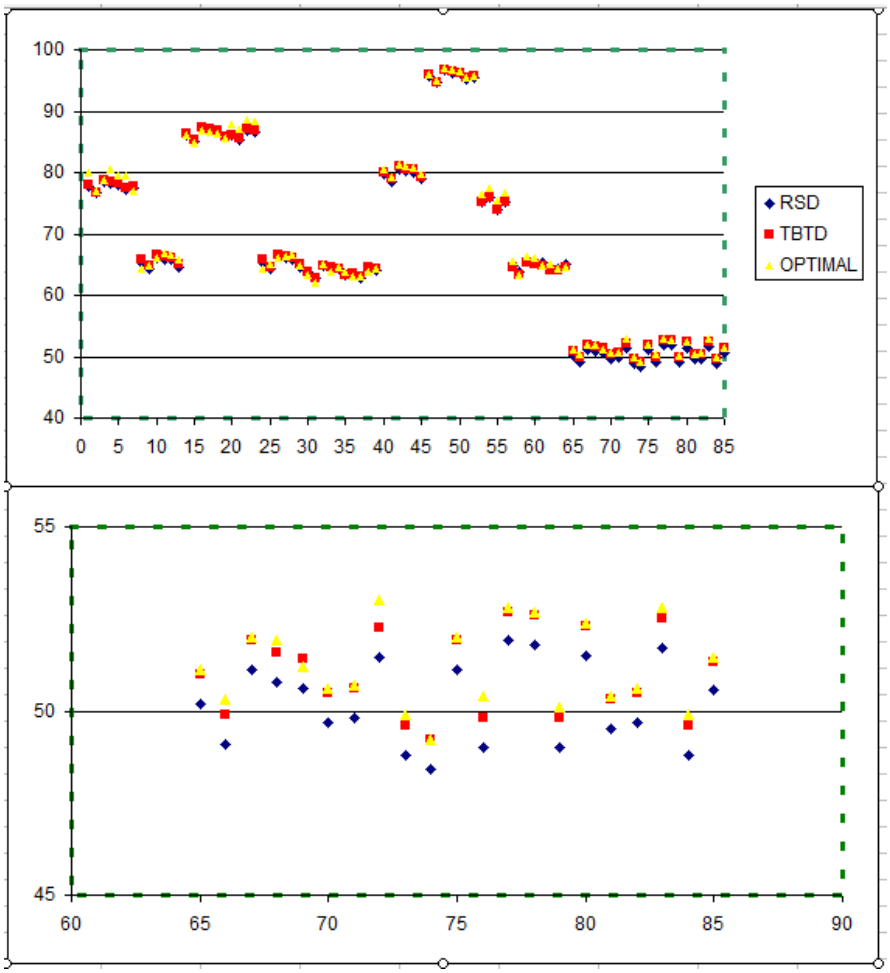

Figure 5.6: Comparison of RSD with optimal algorithm.

An unbalanced curve reveals a situation of dominance of one method over the other, with insights on the intensity of the domination and on the size of the region of dominance [81].

The curves below the cutoff line (at 100) shows that the RSD method is dominated by the other methods and, and the curves above the cutoff line shows that it outperforms the other algorithms. Figure 5.3 show that the TBTD curve is almost flat, with about 70 percent of the attributes having exactly the same performances of the RSD method. However, 30 percent of the attributes demonstrate a slightly better performance of TBTD. The four other most accurate fisher, mdlpc, fusbin and modl methods exhibit balanced but slightly dissymmetric curves: they dominate the RSD method in about 5 to 10 percent

of the attributes and are dominated in about 60 percent of the attributes. Compared to the MDLPC method as shown in figure 5.4, the RSD method is between 0 and 3 percent more accurate in about 70 percent of the attributes, and 0 to 3 percent more accurate in 60 percent attributes in the case of fisher. Among the other methods Chimerge is dominated by RSD in about 95 percent of the attributes but significantly, has a 10 to 12 percent more accuracy in 15 percent of attributes.

5.3.1.1 Comparison Summary

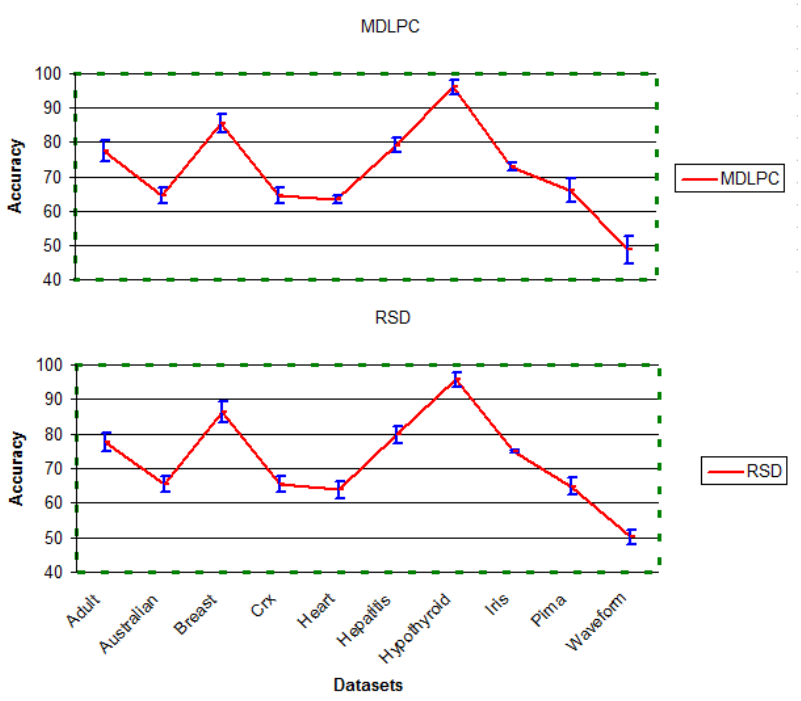

Figure 5.7: Comparison of accuracy and standard deviation between RSD and MDLPC methods.

Instead of comparing the mean accuracy of all the methods, we measure the critical area t^* of the difference of each method. This method has been previously used in [31]. The methods are compared two by two according to the following statistical procedure. Let u and v be two methods to compare. We form the difference Γ_{uv} between the rates of well ordered

elements of the methods u and v. This difference is a random attribute which is roughly normal with parameters (μ, σ). We conclude that u is better than v if μ is significantly superior to 0. We have n = 85 * 10 observations. The estimated mean value μ and mean standard deviation σ are:

$$\mu_{uv} = \frac{1}{850} \sum_{j=1,s=1}^{11,21} \gamma_{js}^{uv} \text{ ; where, } \gamma_{js}^{uv} = \gamma_{js}^{u} - \gamma_{js}^{v}$$

$$\sigma_{uv} = \sqrt{\frac{1}{850} \sum_{j=1,s=1}^{11,21} \gamma_{js}^{uv} - \mu_{uv}} \text{ and } t^* \text{ is: } t^* = \frac{\mu_{uv}}{\sigma_{uv}/\sqrt{n}} > t_{1-\alpha}$$

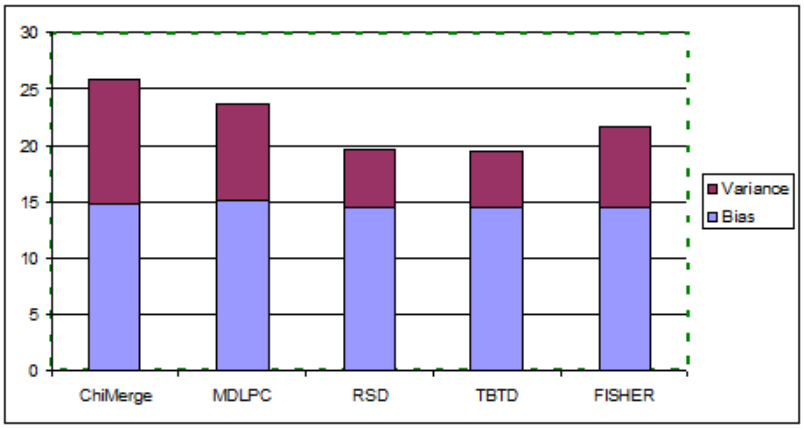

Figure 5.8: Comparison of the bias/variance decomposition between all the methods.

with $t_{1-\alpha}$ the critical value at the rate α of a Student's law with $(n-1)$ degrees of freedom. Since, n is large, we have for $\alpha = 0.05$, $t_{1-\alpha/2} = 1.96$. The computed t^* results are reported in figure 5.5. Positive values of t^* indicate that the method in the row is better than the method in the column. The table shows the performace of Chi-Merge and Bgain methods whose results are the worst among all the methods. MDLPC, FUSBIN, MODL and DBD perform better and have relatively smaller differences. Fisher's algorithm with Fusinter criteria goes a bit further in improving the accuracy, while RDD, RSD and TBTD significantly, report much better results.

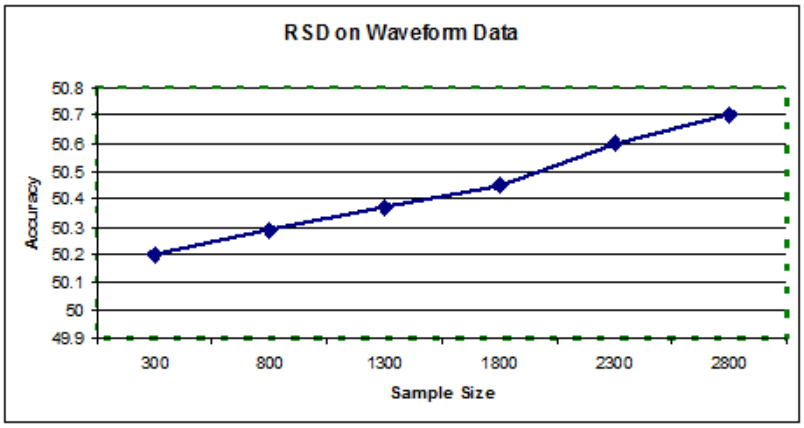

Figure 5.9: Evolution of accuracy in RSD with learning sample size.

5.3.1.2 Quasi-Optimal Discretization

We would like to see how far away our solution lies from the optimal solution or how many times the obtained discretization points miss the global optima. For this purpose, we apply the Fisher optimal algorithm's implementation of the Fusinter criterion on the whole population (entire dataset). This is different from previous experiments where we used the optimal algorithm on smaller samples. Figure 5.6 (top) plots the prediction rates for each of the 85 attributes of RSD, TBTD and the optimal algorithm and then (bottom) magnifies the result of the waveform dataset. It shows a small difference of accuracy of the optimal method and RSD, but a negligible difference from TBTD which only misses the global optima slightly for about 30 percent of the attributes.

5.3.1.3 Bias/Variance Analysis

By building a discretization point distribution by bootstrap and using techniques such as smoothing and averaging we try to lower the discretization variance as previously discussed. Thus, here we plot the accuracy and standard deviation bars for MDLPC and RSD for each of the ten datasets shown in figure 5.7. As excepted the deviation from the mean accuracy is much larger in the case of MDLPC than for RSD. This different is significant in the waveform and pima datasets which have noisy attributes where MDLPC exhibits a

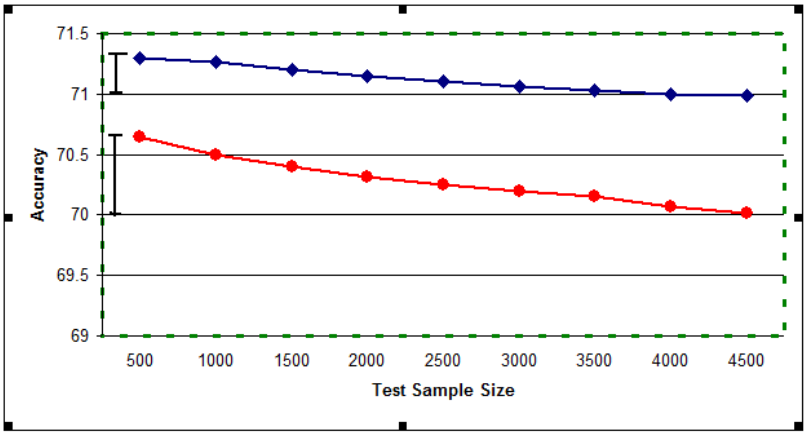

Figure 5.10: Evolution of accuracy in RSD with test sample size.

standard deviation of around 8 compared to 2.3 for RSD.

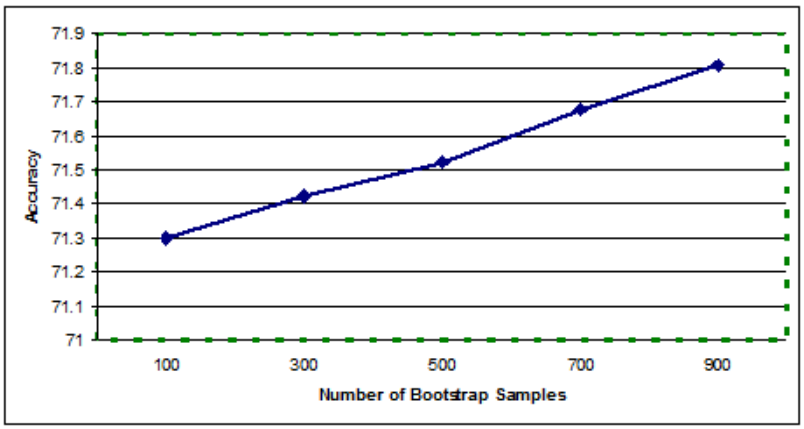

Figure 5.11: Evolution of accuracy in RSD with number of bootstraps.

In figure 5.8 we illustrate the bias/variance decomposition of the chosen methods from all the trials comprising of the geometric mean of all the datasets. It demonstrates a small difference in bias between all the methods, in fact the different of bias error between Fisher and our techniques is negligible. The main difference arises in the production of error caused due to variance. As shown in figure 5.8 both RSD and TBTD. In comparison to the original

MDLPC method the variance is reduced to half of that in both RSD and TBTD schemes.

5.3.1.4 Other Factors Effecting Accuracy

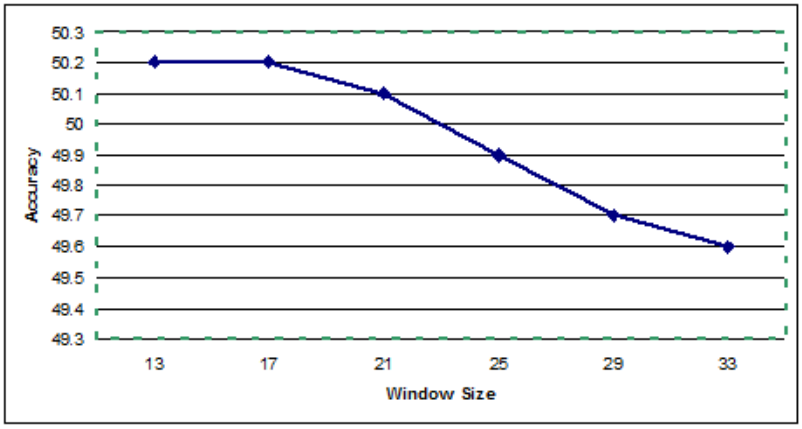

Figure 5.12: Evolution of accuracy in RSD with increasing window size.

In this subsection we discuss other factors that effect the accuracy of discretizations, particularly in the case of RSD technique. Figure 5.9 shows us a steady but a small increase in accuracy of our RSD technique with respect to the sample size measured over the 21 attributes of the waveform dataset. However, figure 5.10 shows an interesting behavior of the RSD method. It compares the accuracy of RSD with MDLPC with respect to the size of the test set in the waveform dataset. Upon examination we can see that the effect of decreasing accuracy with increase in the size of the test data is much less perceived in RSD as compared to MDLPC. This behavior is present due to the fact that even on small samples, RSD builds a reliable discretization point distribution. Overall, MDLPC registers a 7 percent decrease in accuracy as compared to 3 percent in RSD upon increasing test set size.

Since, our technique is based upon bootstrap sample, we study the effect of the number of bootstrap samples generated on the prediction accuracy. Figure 5.11 demonstrates this effect on the mean accuracy from all the data sets. There is a linear improvement in accuracy but the increase from 100 to 1000 bootstrap samples is just about 0.6 percent.

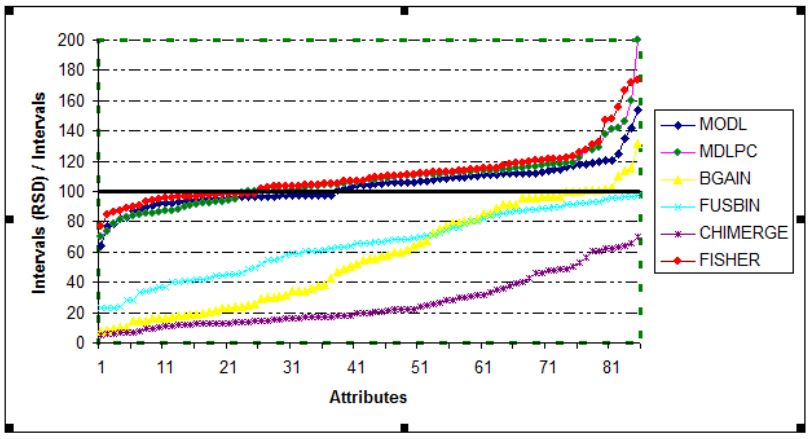

Figure 5.13: Comparison of the Intervals between all the methods.

Since, the RSD technique uses a moving average filter of size ws to smoothen the discretization point distribution curve, the size of ws does have an effect on the discretization quality. Figure 5.12 illustrates this effect on the accuracy while, its effect on the number of intervals is explained in the next section. The curve clearly shows the decrease in accuracy with an increase in window size for the waveform data set. The increase in window size improves the generalization effect of the obtained discretization hence, the accuracy suffers. But, the curve is not linear and seem to fall suddenly after $ws = 20$.

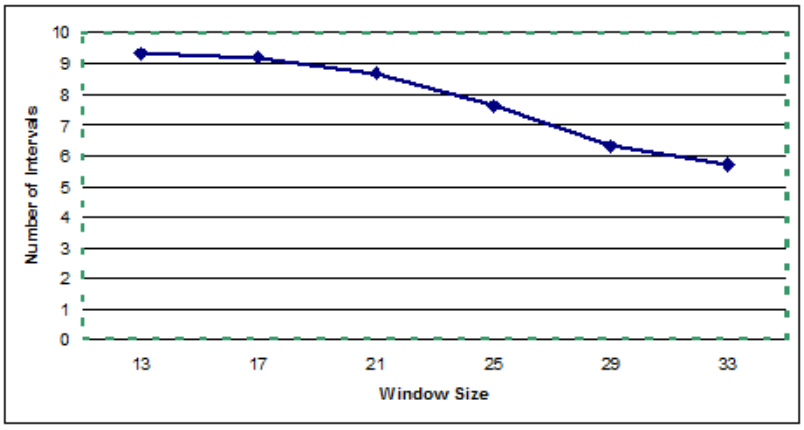

Figure 5.14: Evolution of number of intervals in RSD with increasing window size.

5.3.2 Complexity - Number of Intervals

The number of intervals or partitions play a vital role in the generalization and prediction variance of the obtained discretization. It is also responsible of increased complexity which is discussed in chapter 3. Figure 5.13 shows the repartition function of the relative differences of number of intervals between the RSD method and the other discretization methods. The curves above the cutoff line (at 100) shows that the RSD method is dominated by the other methods and, and the curves below the cutoff line shows that it outperforms the other algorithms. It shows that the RSD dominates throughout the Chimerge and the fusbin curve but the later is much closer to RSD cutoff line. In the case of Bgain, RSD dominates significantly for about 85 percent of the attributes but is inferior in the rest of 15 percent. The modl, mdlpc and fisher have almost similar curves which are mostly flat for almost 70 percent of the time. RSD dominates for about 10 percent attributes is dominated significantly in the rest of the 20 percent of attributes.

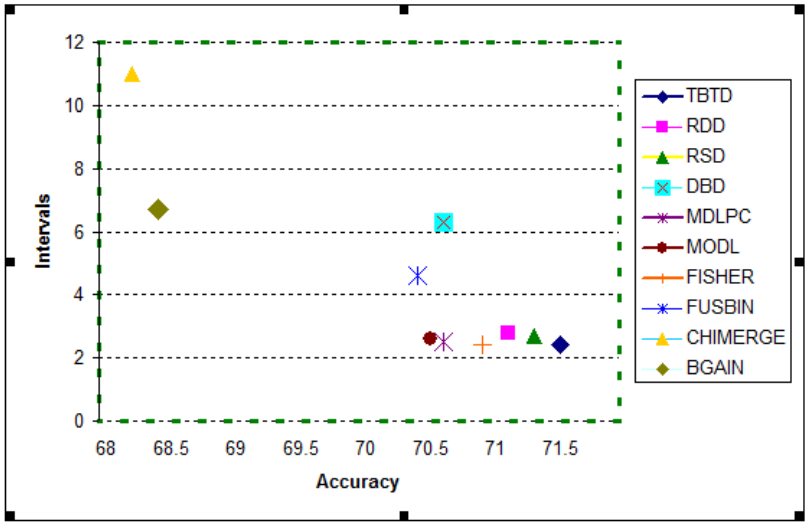

Figure 5.15: Bi-criteria evaluation of the methods for the accuracy and number of intervals, using datasets geometric means.

As discussed previously, window size plays a significant role in both the accuracy and the number of intervals. This fact is illustrated in figure 5.14 where the number of intervals

obtained decreases with growing window size. The number of intervals decrease gradually and then starts to steepen.

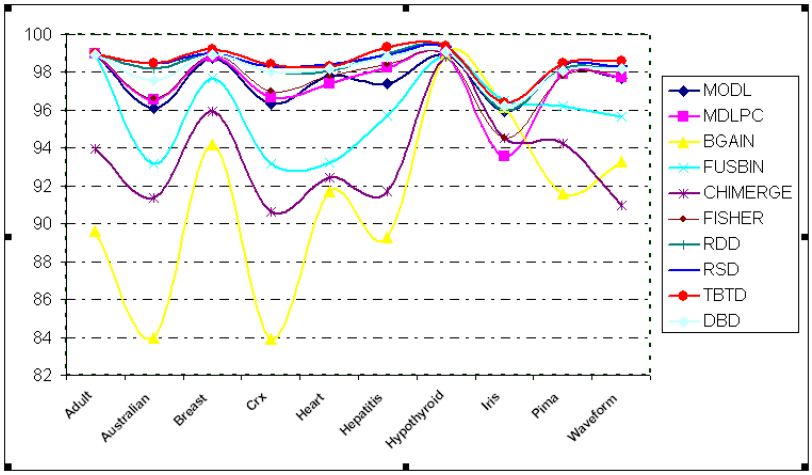

Figure 5.16: Comparison of Robustness between all methods.

5.3.2.1 Bi-criteria Evaluation

A multi-criteria analysis scheme has been used in [81] in order to provide a summary of the comparison of methods when multi criteria are present. In this a solution dominates (or is non-inferior to) another one if it is better for all criteria. Here any improvement of one of the criteria causes a deterioration on another criterion. In order to analyze both the accuracy and robustness results, we report the dataset geometric means on a two-criteria plan in Figure 5a, with the accuracy on the x-coordinate and the robustness on the y-coordinate. Similarly, we report the accuracy and the number of intervals in Figure 5b. Each point in these figures represents the summary of all the experiments. The multi-criteria figures are thus reliable and informative: they allow us to clearly differentiate the behavior of almost all the methods.

Accuracy is certainly the most important parameter to distinguish a discretization method so we have grouped it in both the analysis of figure 5.15 and 5.17. In figure 5.14 we put RDD, RSD, TBTD and fisher's optimal algorithm in the same group on the right hand side

having both better accuracy and comparable number of intervals. Then, come MDLPC and Modl which have lesser accuracy but same number of intervals. The third category comprises of Fusbin and DBD and then finally Bgain and Chimerge have the worst accuracies and number of intervals.

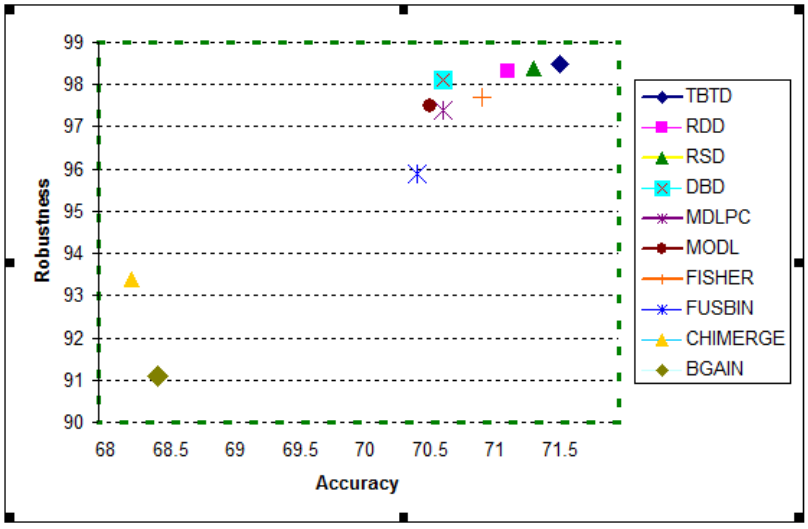

Figure 5.17: Bi-criteria evaluation of the methods for the accuracy and robustness, using datasets geometric means.

5.3.3 Robustness

The robustness is an interesting criterion that allows to estimate whether the performance on the sample train data is a good prediction of the performance on the whole data set (Population). The higher is the robustness, the most accurate will be a ranking of attributes based on their sample accuracy. This can be critical in the case of classifiers such as decision trees that incorporate an attribute selection method to build the next node of the tree. Figure 5.16 plots the curves of the mean robustness for each dataset for all the methods. It can be seen that all our techniques seem to exhibit high robustness in almost all the data sets while MDLPC, MODL and Fisher lie just below them. Bgain and chimerge are the least robust.

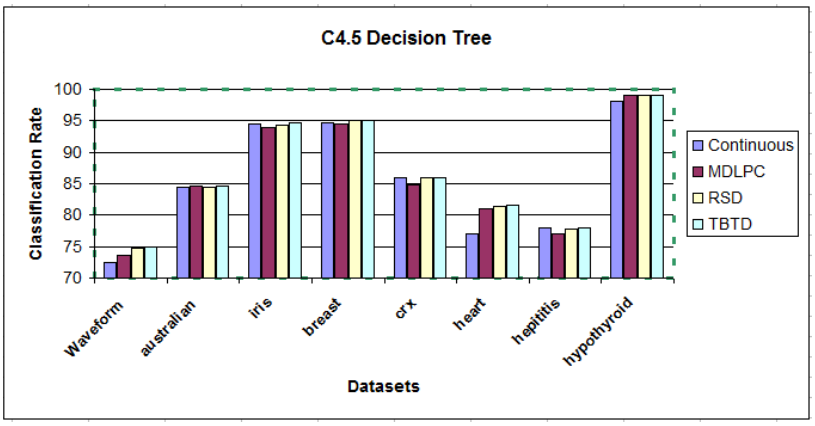

Figure 5.18: Comparison of decision tree accuracy using all the methods.

5.3.3.1 Bi-criteria Evaluation

To sum up this criteria we do a bicriteria analysis of robustness with accuracy which is shown in figure 5.17. It shows the three methods RSD, TBTD and RDD on the upper right hand corner with the best performace. Next, comes Fisher's algorithm and then the three methods in close proximity i.e. MDLPC, Modl and DBD, while on the bottom left corner are Bgain and Chimerge.

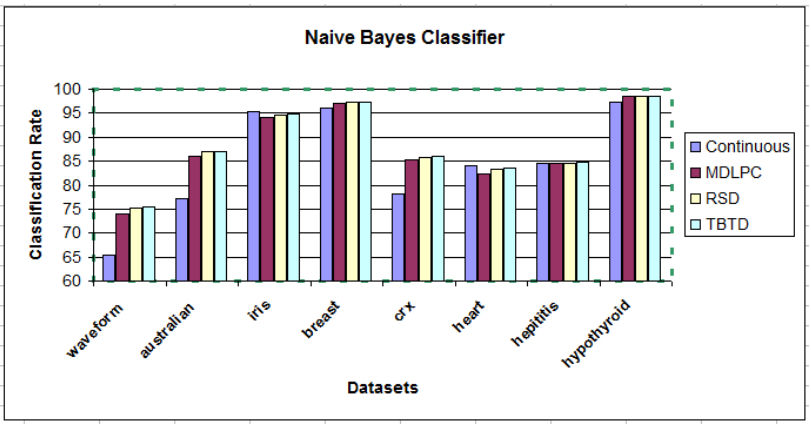

Figure 5.19: Comparison of naive-bayes accuracy using all the methods.

5.3.4 Classification Accuracy with Discretization as a Pre-Processing Step

In this section we use discretization as a preprocessing step for classifiers such as decision trees and naive-bayes classifiers to provide additional insights about the contribution of various discretization methods, significantly our approaches. We compare in terms of classification rate, RSD and TBTD with MDLPC and no discretization i.e. generating a decision tree without applying any discretization procedure. Figure 5.18 demonstrates the performance of the C4.5 decision tree on each data set using the above mentioned four techniques. RSD and TBTD give almost similar results but there seem to be a significant difference in comparison when no pre-processing step is applied. However, the australian and hypothyroid datasets display almost similar classification rates. Waveform and heart data sets get the best out of RSD and TBTD.

Next, we do the same experiments using the naive bayes classifier. Here instead of using continuous data we assume a gaussian prior distribution model. Figure 5.19 shows a significant improvement in classification for the waveform, australian, crx data sets while small improvements in the others. In the case of hepatitus data set the techniques seem to align.

5.3.5 Time Complexity

The evaluated discretization methods have the same computational complexity of $O(nlog(n))$. They first sort the attribute values and identify boundary instances in a preprocessing step. The time efficiency of the methods mainly relies on the search direction (top-down or bottom-up), the simplicity of the mathematical criterion. The optimal algorithm has a complexity of $O(N^3)$ while RDD and RSD depend on the number of bootstrap samples B in addition to small post-processing that includes the extraction of discretization points from the distribution. Thus, their complexity becomes $B \times O(nlog(n))$. In the case of TBTD an additional cost is added which consists of generating B bootstrap samples at each node instead of generating them once for the sample of the root node (as in RSD and RDD). While, the complexity of DBD is almost linear $B \times O(N)$. Thus, the comparison of the time complexities is as follows:

$$DBD < (MDLPC < Bgain < Chimerge < Modl) < RDD < RSD < TBTD < Fisher$$

Although here, we argue that there is a trade-off between time complexity and quality. But with vast improvements in computing speeds, we argue that quality could be a much valuable commodity.

5.4 Conclusions of this Part

The learning sample is an approximation of the whole population, so the optimal discretization built on a single sample set is not necessarily the global optimal one. Our resampling based approaches tend to give a better discretization estimate in terms of achieving better discretization quality. RDD, RSD and TBTD methods improved the discretization variance, robustness, prediction accuracies and provided smaller number of intervals. Thus, arrived nearer to a global optimal solution (FisherŠs optimal algorithm). In the case of DBD the number of intervals were much higher than the other three methods and thus, also exhibit larger variance. They worked better for naive-bayes classifiers than decision trees. Whereas the other methods worked better in the case of decision trees because they used MDLP discretization as the base discretizer, which is known to work better for decision trees than naive-bayes (see chapter 3).

Among other methods except for Chi-Merge and Balancedgain, the other methods provide small variations in terms of prediction rates. MDLPC performs the best in terms of number of intervals and time complexity, MODL and Fusinter were not far behind. Fisher's method is the most expensive computationally while our DBD approach is the least. However, RSD and TBTD are more expensive than all the methods (except Fisher), but this phenomenon can be controlled by the number of bootstrap samples. We argue that the availability of high computational power these days can help us in focusing more on achieving better discretization quality.

These techniques have been applied in the context of fuzzy or soft discretization [138] in decision trees which is discussed in the coming chapters.