Unsupervised Discretization using Tree-based Density Estimation

Gabi Schmidberger and Eibe Frank

Department of Computer Science
University of Waikato
Hamilton, New Zealand
{gabi, eibe}@cs.waikato.ac.nz

Abstract. This paper presents an unsupervised discretization method that performs density estimation for univariate data. The subintervals that the discretization produces can be used as the bins of a histogram. Histograms are a very simple and broadly understood means for displaying data, and our method automatically adapts bin widths to the data. It uses the log-likelihood as the scoring function to select cut points and the cross-validated log-likelihood to select the number of intervals. We compare this method with equal-width discretization where we also select the number of bins using the cross-validated log-likelihood and with equal-frequency discretization.

1 Introduction

Discretization is applied whenever continuous data needs to be transformed into discrete data. We consider data that is organized into instances with a fixed number of attributes. Some or all of the attributes might be continuous. To discretize a continuous attribute the range of its values is divided into intervals. The attribute values are substituted by an identifier for each bin. Note that the new attribute is actually not categorical but ordered [1]. In this paper we consider the problem of unsupervised discretization, where the discretization is based on the distribution of attribute values alone and there are no class labels. Moreover, we consider univariate discretization: our method only considers the attribute to be discretized, not the values of other attributes.

Our algorithm treats unsupervised discretization as piece-wise constant density estimation. The intervals gained from the discretization can be used to draw a histogram or used to pre-process the data for another data mining scheme. In the former case, the height of each bin h is the density that is computed from the bin width w_i, the number of instances n_i that fall into that bin, and the total number of instances N in the dataset:

$$h = \frac{n_i}{w_i * N} \tag{1}$$

The paper is organized as follows. In Section 2 we discuss non-parametric density estimation, of which our histogram estimator is a special case. We compare our discretization method to equal-width discretization (and variants) and equal-frequency discretization. These methods are summarized in Section 3. In Section 4 we discuss the cross-validated log-likelihood, which we use as the model selection criterion to choose an appropriate number of bins. Our method is explained in detail in Section 5. An experimental comparison is presented in Section 6. Section 7 has some concluding remarks.

2 Non-parametric Density Estimation

Density estimation, parametric or non-parametric, is about constructing an estimated density function from some given data. Parametric density estimation assumes that the data has a density function that is of a known family of distributions. For example, the distributions of the normal or Gaussian model have the parameters μ for the mean and σ^2 for the variance. The parametric method has to find the parameters for the best fit to the data. However, practical applications showed that there is often data that cannot be fit well enough with parametric methods, so non-parametric methods have been developed that work without the assumption of one of these specific distributions and fit more complex and flexible models to the data [2].

Non-parametric density estimation suffers from two major problems: the curse of dimensionality and finding a good smoothing parameter [3]. Since we work in the univariate case the curse of dimensionality does not matter in our application. What still remains is the problem of finding a good smoothing parameter. The smoothing parameter for histograms is the number of bins. If the histogram has many bins, the density curve will show many details; if the bins get fewer, the density curve will appear smoother. The question is how much is enough detail. Our method automatically finds an appropriate bin width based on cross-validation and the width is not constant for the whole range but adapts to the data (i.e. the bin width varies locally).

Histograms have been widely used because they generate an easily understandable representation of the data. They represent the density function as a piece-wise constant function. Alternative methods of non-parametric density estimation are kernel estimators, which represent the data with a smooth function [2]. Although kernel density estimators avoid the discontinuities present in histograms, they cannot be used to summarize the data in a concise form and are more difficult to explain to the non-expert.

A further disadvantage of the kernel method is its computational complexity. Assuming all kernels contribute to the density, computing the density for a test instance requires time linear in the number of training instances. In contrast a binary search on the bins is sufficient in a histogram, and the time complexity is logarithmic in the number of bins. This is particular relevant for large datasets. Kernel density estimation is a lazy method. Our method is an eager method that requires more effort at training time.

3 Existing Unsupervised Discretization Methods

We compare our method with two well-known unsupervised discretization methods: equal-width discretization and equal-frequency discretization. Equal-width discretization divides the range of the attribute into a fixed number of intervals of equal length. The user specifies the number of intervals as a parameter. A variant of this method, which we also compare against, selects the number of intervals using the cross-validated log-likelihood. This variant is implemented in Weka [4].

For equal-width histograms it is not only important to select the number of intervals but also the origin of the bins [2]. The origin is found by shifting the grid by a part of the actual bin width (e.g. one 10^{th} of it), and selecting the best one of these shifts. We implemented this by using the cross-validated log-likelihood to select the origin and the number of bins.

Equal-frequency discretization also has a fixed number of intervals, but the intervals are chosen so that each one has the same or approximately the same number of instances in it. The number of intervals is determined by the user.

4 Cross-validating the Log-likelihood

Cross-validation is used in machine learning to evaluate the fit of a model to the real distribution. It is generally applied to classification or regression problems but it can also be applied to clustering [5]. The idea is to split the dataset into n equal-sized folds and repeat the training process n times using $n-1$ folds for training and the remainder for testing. This is done with every parameter value and the best value is chosen to build the final model based on the full training set. Various scoring functions are used to decide which parameter value is the best. We use the log-likelihood of the histogram, which is also used for fitting mixtures of normal distributions for clusters.

The log-likelihood is a commonly used measurement to evaluate density estimators [2]. It measures how likely the model is, given the data. Choosing the model that maximizes the likelihood on the training data results in overfitting, analogue to the classification or regression case. Cross-validation gives an (almost) unbiased estimate of performance on the true distribution and is a more suitable criterion for determining model complexity.

Leave-one-out cross-validation can be applied in the case of equal-width discretization because the log-likelihood on each test instance can be easily computed as the bins stay fixed. In our new discretization method the location of each cut point can change with one instance removed, making the leave-one-out method too expensive. Hence we use 10-fold cross-validation instead.

Let n_i be the number of training instances in bin i, n_{i-test} the number of instances of the test set that fall into this bin, w_i the bin width, and N the total number of training instances. Then the log-likelihood L on the test data is:

$$L = \sum_i n_{i-test} * log \frac{n_i}{w_i * N} \tag{2}$$

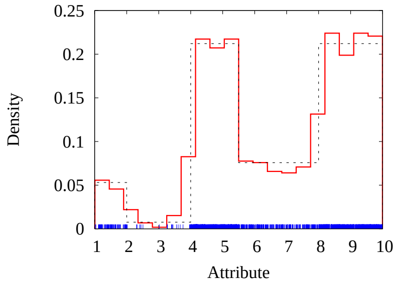

Fig. 1. Equal-width method with 20 bins.

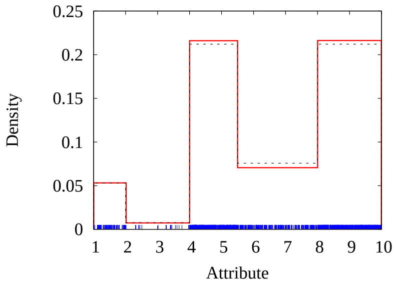

Fig. 2. Our TUBE-Method chose 5 bins of varying length.

There is one problem: empty bins. If n_i equals zero the logarithm is undefined. To solve this problem we spread a single instance over the whole range of the data by adding a part of the instance to each bin that is equivalent to the relative width of the bin. Assume W denotes the total length of the range. Then the above formula becomes:

$$L = \sum_i n_{i-test} * log \frac{n_i + \frac{w_i}{W}}{w_i * (N+1)} \tag{3}$$

5 Tree-based Unsupervised Discretization

The most common and simplest way of making histograms is the equal-width method. The range is divided into subranges or bins of equal-width. In contrast to this, our new algorithm divides the range of an attribute into intervals of varying length. The goal is to cut the range in such a way that intervals are defined that exhibit uniform density. Of course, in practical problems the true underlying density will not really be uniform in any subrange but the most significant changes in density should be picked up and result in separate intervals. Figure 1 shows an equal-width estimator for a simple artificial dataset. Figure 2 shows the density function generated with our discretization method. In both figures the "true" density (the density function that was used to generate the data) is plotted with a dotted line. The training data is shown as vertical bars at the bottom.

We call our method TUBE (Tree-based Unsupervised Bin Estimator), because it uses a tree-based algorithm to determine the cut points. More specifically, it builds a density estimation tree in a top-down fashion. Each node defines one split point. In the following we describe how a locally optimum split point can be found for a subset of data. Then we describe how the tree is built and pruned and what can be done about the problem of "small" cuts.

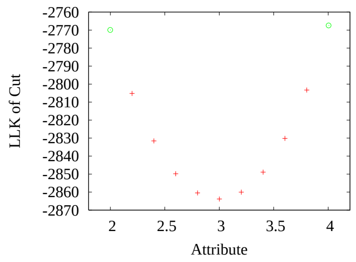

Fig. 3. The log-likelihood is minimized between instances.

5.1 Where to Cut

The quality of the density estimation is measured by the log-likelihood. We choose the split point that maximizes the likelihood based on the training data and the ranges at the current node of our density estimation tree:

$$L = n_{left} * log \frac{n_{left}}{w_{left} * N} + n_{right} * log \frac{n_{right}}{w_{right} * N} \quad (4)$$

Here, n_{left} is the number of instances in the left subrange and w_{left} its width. The quantities for the right subrange are defined accordingly. In contrast to decision tree learning [6], every training instance defines two potential maximum likelihood cut points. The cut is made at the instance (and not in-between instances as in the case of classification or regression trees) and includes the instance in either (a) the left or (b) the right subset. This is because the log-likelihood of a division into two bins has a local minimum if the cut point is set in-between two points. It attains a local maximum at the points. The diagram in Figure 3 shows the log-likelihood of cut points at two values of an attribute (circles) and at nine points in-between the values (crosses). The log-likelihood is maximized at the instance values. Hence we cut at the values of the training instances and consider including a point in either the right or the left subset—i.e. we consider the interval boundaries $.., x)[x, ..$ and $..x](x, ..$ for an instance x.

Note that this causes problems when the data is very discontinuous and has a "spike" in its range (i.e. several identical training instances at the minimum or maximum). The estimated density would be infinity because the range would be zero. Therefore our implementation does not actually cut at the instance value itself but add, or substract, a small value (we used 10^{-4} in our experiments).

5.2 Building the Tree

The selection of k cut points can be seen as a search through a resolution space for the optimal solution. A well-known search method is the divide-and-conquer method that decision trees use. On numeric attributes this method finds the locally optimal binary split and repeats the process recursively in all subranges until a stopping criterion is met. This is a greedy search that does not find

```
maxNumBins                        = [find optimal number of splits];
numSplits                         = 0;
splitPriorityQueue                = empty;

firstBin                  = new Bin(bin that contains the whole attribute range);
fringe                    = [initialize with (firstBin)];

REPEAT {
    FOR (bin = all bins in fringe) {
        split = bin.[find best split in the range of this bin];
        splitPriorityQueue.[add (split)];
        fringe.[delete (bin)];
    }
    nextBestSplit = splitPriorityQueue.[give best split in queue];
    newBinLeft, newBinRight =
                nextBestSplit.[perform split on its bin and replace the bin with the
                                    two new bins newBinLeft and newBinRight];
    numSplits++;
    fringe.[add the two new bins (newBinLeft, newBinRight)];
} UNTIL (numSplits == maxNumBins - 1);
```

Fig. 4. Pseudo code for the tree building algorithm.

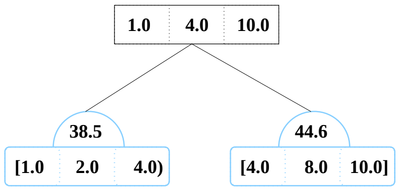

Fig. 5. Tree after the first cut.

an optimal global division but a division that is a computational inexpensive estimate. We apply this method to unsupervised discretization using best-first node expansion. The pseudo code of our algorithm is shown in Figure 4.

In the following we present an example using the dataset from Figures 1 and 2. First the best cut point is found in the whole range and two new bins are formed. Within the two subranges two new locally optimal cut points are searched for. Both splits are evaluated and a log-likelihood for the division into the resulting three bins is computed for both possible splits. Figure 5 shows the discretization tree corresponding to this situation. The root node represents the first cut and the two leaf nodes represent the next two possible cuts.[1]

Each node represents a subrange, the root node the whole range. The variables written to the left and right side of the square corresponding to a node represent the minimum and maximum of the subrange. The overall minimum and maximum of this example dataset are 1.0 and 10.0. Each leaf node represents a bin and the minimum exhibits a "[" if the minimum value itself is part

[1] All values are rounded.

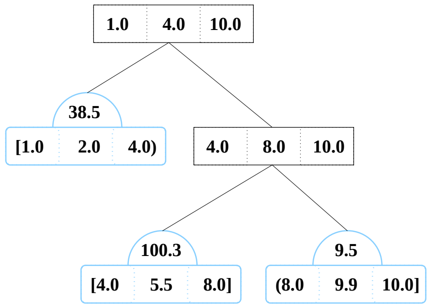

Fig. 6. Tree after the second cut.

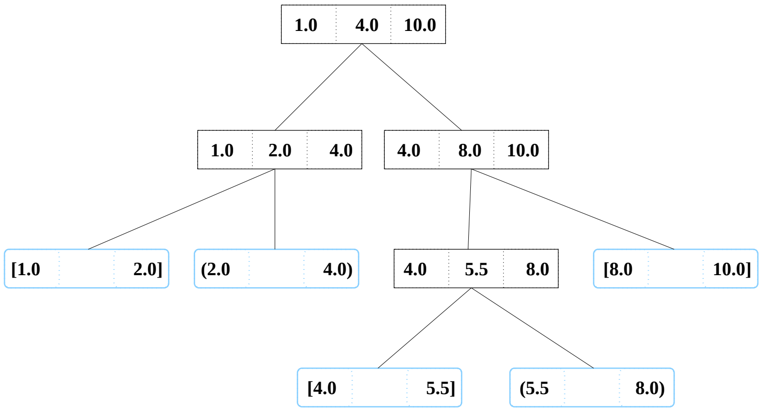

Fig. 7. Finalized tree.

of the bin and a "(" if it is part of the next bin. The notation for the maximum is analogue. The variable written in the middle of the node represents the cut point.

The whole range is first cut at the value 4.0. The next possible cut point is either 2.0 or 8.0. These would split the dataset into the subranges [1.0:2.0] [2.0:4.0] [4.0:10.0] or [1.0:4.0] [4.0:8.0] [8.0:10.0] respectively. The gain in log-likelihood for each of the two possible divisions is written in the half-circle over the not-yet-exercised cuts. The cut at 2.0 results in a log-likelihood gain of 38.5 computed based on Formula 4, the cut at 8.0 has a log-likelihood gain of 44.6. Among the possible cuts the one with the largest gain in log-likelihood is selected, which in this case is the cut at 8.0. Figure 6 shows the state of the dicretization tree after two cuts.

After the cut at 8.0 is performed, two new bins are generated, and in each of them a new possible cut is searched for. These cuts are 5.5 and 9.9, with log-likelihood gains of 100.3 and 9.5 respectively. So for the third cut there is a choice between three cuts (including the cut at 2.0) and the next one chosen would be 5.5.

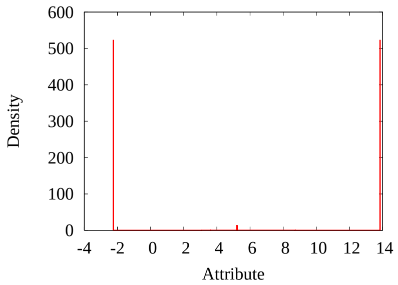

Fig. 8. Distorted histogram due to small cuts.

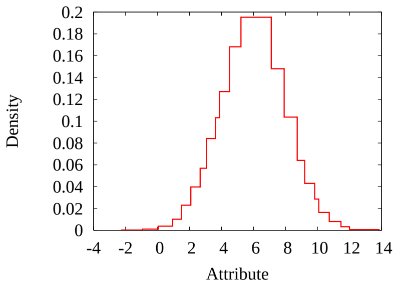

Fig. 9. Small cuts eliminated with heuristic.

After four cuts our discretization tree learning algorithm decides to stop. The stopping criterion will be explained in Section 5.3. Figure 7 shows the final discretization tree. The resulting histogram is the one shown at the beginning of this section in Figure 2. In the final tree each leaf node represents a bin of the histogram. Each internal node represents a cut.

5.3 The Stopping Criterion

The third and last part of our algorithm is the stopping criterion. Based on the likelihood on the training data, the algorithm would not stop cutting until all subranges contain a single value (i.e. it would overfit). The stopping criterion sets the maximal number of cut points and prevents overfitting.

We use the 10-fold cross-validated log-likelihood to find an appropriate number. We start with zero and increase the maximal number of cut points in increments of one. This can be implemented efficiently: to find k cut points, one can use the division into $k-1$ cut points and add one more. By default the algorithm iterates up to $N-1$ as the maximal number of cut points (i.e. the cross-validated log-likelihood is computed for all trees with 1 up to $N-1$ cut points). For each of the $N-1$ iterations the algorithm computes the average log-likelihood over the test folds and from this the number of splits that exhibits the maximum value is chosen.

In our above example the cross-validated log-likelihood curve has its maximum at four cut points and therefore four cuts have been performed. Note that this method involves growing a density estimation tree eleven times: first once for each of the ten training folds, and finally for the full dataset based on the chosen number of cut points. The time complexity of the discretization algorithm is $O(NlogN)$.

5.4 A Problem: Small Cuts

The log-likelihood criterion used to find the cut point in a range can be unstable. Sometimes a cut is found at the border of a subrange that contains very few

Table 1. 464 numeric attributes from UCI datasets and their levels of uniqueness.

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

instances and is very small. This can lead to a very high density value and the criterion decides to cut.

Hence we implemented a heuristic that avoids these small cuts in most cases. More specifically, we disallow cuts that are smaller than 0.1 percent of the whole range of the data and set the minimum number of instances to $\sqrt{0.1 * N}$. Figure 8 shows a strongly distorted histogram of a normal density that is due to two small cuts that have very high density. This was created by our method without using the heuristic. The same dataset is used in Figure 9, where the small cuts have been avoided using the heuristic.

6 Evaluation

We evaluated the TUBE discretization method using numeric attributes from 21 UCI datasets [7]. The algorithm works on univariate numeric data, and thus the numeric attributes of the UCI datasets have been extracted and converted into 464 one-attribute datasets.

A surprising finding was that many of these numeric attributes have a low uniqueness in their values. Low uniqueness means that they have many instances with the same value. Table 1 lists the number of attributes sorted into columns according to their level of uniqueness (e.g. $[0-20)$ means that the percentage of unique values is between 0 and 20). The table also shows the UCI datasets the attributes have been extracted from and the number of instances.

These datasets are used to test how well TUBE discretization estimates the true density. The density estimates that are generated are evaluated using 10x10-fold cross-validation, measuring the log-likelihood on the test data. Note that this "outer" cross-validation was performed in addition to the "inner" cross-validation used to select the number of cut points.

Our new discretization method (TUBE) is compared against equal-width discretization with 10 bins (EW-10), equal-width with cross-validation for the number of bins (EWcvB), equal-width with cross-validation for the origin of the bins and the number of bins (EWcvBO), and equal-frequency discretization with 10 bins (EF-10). The equal-frequency method could not produce useful models for the attributes with uniqueness lower than 20 and has therefore been left out in that category. TUBE, EWcvB and EWcvBO were all run with the maximum bin number set to 100.

6.1 Evaluating the Fit to the True Distribution

Table 2 lists the summary of the comparison. Each value in the table is the percentage of all attributes in that uniqueness category for which TUBE was significantly better, equal or worse respectively based on the corrected resampled t-test [8]. In almost all cases our method is at least as good as the other methods and shows especially good results in cases with low uniqueness and some cases of high uniqueness. We have analyzed the corresponding attributes and they show that TUBE is generally better when attributes exhibit discontinuities in their distributions.

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

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

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

6.2 Comparing the Number of Bins

Table 3 shows a comparison of the number of bins generated by the different methods. A smaller number of bins gives histograms that are easier to under-

Table 2. Comparison of the density estimation results. Result of paired t-test based on cross-validated log-likelihood.

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

stand and analyze. In the category 80 percent and higher the TUBE discretization generates a significantly smaller number of bins than the other methods. Whenever it produces more bins this appears to result in a better fit to the data.

7 Conclusion

TUBE discretization provides a good algorithm for density estimation with histograms. The density estimation of very discontinuous data is often difficult. Our results show that TUBE outperforms equal-width and equal-frequency discretization on discontinuous attributes. It finds and can represent "spikes" in the density function that are caused by discrete data (many instances with the same value) and can reliably detect empty areas present in the value range.

On truly continuous data the method provides a discretization that represents the data as well as the other methods but with fewer bins and hence gives a clearer picture of areas of different density. A possible application of our method would be density estimation for classification in Naive Bayes [9]. We plan to investigate this application in future work.

Table 3. Comparison of the number of bins.

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

References

1. Eibe Frank and Ian H. Witten. Making better use of global discretization. *Proc. of the Sixteenth International Conference on Machine Learning*, pages 115–123, 1999.
2. B.W.Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.
3. D.W.Scott. *Multivariate Density Estimation*. John Wiley & Sons, 1992.
4. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2000.
5. P. Smyth. Model selection for probabilistic clustering using cross-validated likelihood. *Staistics and Computing*, pages 63–72, 2000.
6. J.R.Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kaufmann, 1993.
7. C.L. Blake S. Hettich and C.J. Merz. UCI repository of machine learning databases, 1998.
8. C. Nadeau and Y. Bengio. Inference for the generalization error. *Machine Learning*, 52:239–281, 2003.
9. Y. Yang and G. I. Webb. Proportional k-interval discretization for naive-bayes classifiers. In *Proceedings of the 12th European Conference on Machine Learning (ECML)*, pages 159–173, Tokyo, 2001.